Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

2016-03

Mapping, awareness, and virtualization network administrator training tool virtualization module

Berndt, Erik W.

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/48563

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL VIRTUALIZATION MODULE

by

Erik W. Berndt

March 2016

Thesis Advisor: John Gibson
Co-Advisor: Alan Shaffer

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL VIRTUALIZATION MODULE			**5. FUNDING NUMBERS**
6. AUTHOR(S) Erik W. Berndt			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Conducting network administration training in an operational tactical network environment introduces a level of risk to the network that is unacceptable to operational commanders. This forces a choice between the readiness of network administrators and the availability of the tactical network as an operational platform. To address this conflict, the Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) architecture has been designed to map and enumerate an operational network in order to create a fully partitioned, faithful virtual copy of that network that can be used safely for network administrator training. A capability does not exist to automatically generate a virtual copy of a network based on a graphical model of that network. This work solved this problem by implementing a prototype of the MAVNATT Virtualization Module, which creates a virtual copy of a mapped operational network. We tested the output of the Virtualization Module against the functional requirements of the MAVNATT system. The prototype implementation successfully integrates a virtualization hypervisor, a network simulator, and a WAN emulator to create a virtual network based on a graph model of an operational network, and demonstrates the ultimate attainability of the MAVNATT architecture's objectives.

14. SUBJECT TERMS virtualization, network administrator training, simulation, MAST, MAVNATT, tactical network			**15. NUMBER OF PAGES** 81
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MAPPING, AWARENESS, AND VIRTUALIZATION NETWORK ADMINISTRATOR TRAINING TOOL VIRTUALIZATION MODULE

Erik W. Berndt
Lieutenant, United States Coast Guard
B.S., Golden Gate University, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: John Gibson
Thesis Advisor

Alan Shaffer
Co-Advisor

Cynthia Irvine
Chair, Cyber Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Conducting network administration training in an operational tactical network environment introduces a level of risk to the network that is unacceptable to operational commanders. This forces a choice between the readiness of network administrators and the availability of the tactical network as an operational platform. To address this conflict, the Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) architecture has been designed to map and enumerate an operational network in order to create a fully partitioned, faithful virtual copy of that network that can be used safely for network administrator training. A capability does not exist to automatically generate a virtual copy of a network based on a graphical model of that network. This work solved this problem by implementing a prototype of the MAVNATT Virtualization Module, which creates a virtual copy of a mapped operational network. We tested the output of the Virtualization Module against the functional requirements of the MAVNATT system. The prototype implementation successfully integrates a virtualization hypervisor, a network simulator, and a WAN emulator to create a virtual network based on a graph model of an operational network, and demonstrates the ultimate attainability of the MAVNATT architecture's objectives.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
- **A. PROBLEM STATEMENT....1**
  - **1. Primary Question....2**
  - **2. Secondary Questions....2**
- **B. ASSUMPTIONS....2**
- **C. OBJECTIVES....3**
- **D. BENEFITS OF STUDY....3**
- **E. ORGANIZATION....3**

II. BACKGROUND AND PREVIOUS WORK....5
- **A. OVERVIEW....5**
- **B. MAPPING MODULE....6**
- **C. AWARENESS MODULE....6**
- **D. VIRTUALIZATION MODULE....7**
  - **1. MAVNATT Virtualization Module Program....7**
  - **2. MAVNATT Virtualization Hypervisor....9**
  - **3. MAVNATT Network Simulator....11**
  - **4. MAVNATT WAN Emulator....13**
  - **5. Integration with the MAST Training Tool....14**
- **E. SUMMARY....16**

III. SYSTEM REQUIREMENTS AND DESIGN....17
- **A. OVERVIEW....17**
- **B. VIRTUALIZATION MODULE PROTOTYPE REQUIREMENTS....17**
- **C. SYSTEM DESIGN....19**
  - **1. GraphML Input File....19**
  - **2. Virtualization Module Program....22**
  - **3. Virtualization Hypervisor....22**
  - **4. Network Simulator....25**
  - **5. Wide Area Network Emulator....26**
  - **6. Physical-to-Virtual Process....28**
    - ***a. P2V Overview....28***
    - ***b. P2V Benefits....29***
    - ***a. P2V Drawbacks....30***
- **D. SUMMARY....31**

IV. SYSTEM IMPLEMENTATION....33

A. OVERVIEW 33
B. COMPONENT IMPLEMENTATION 33
1. Prototype Design 33
2. Virtualization Module Conceptual Network 34
3. GraphML Input File 35
4. Virtualization Module Program 39
5. Virtualization Hypervisor 40
6. Network Simulator 41
7. Wide Area Network Emulator 42
C. SUMMARY 43

V. VIRTUALIZATION MODULE PROTOTYPE TESTING 45
A. OVERVIEW 45
B. UNIT TESTING 45
1. Validation Routine and Network Topology 45
2. VM Cloning 46
3. Topology File Creation 47
4. Network Device Cloning 47
5. Network Partitioning 47
C. SYSTEM RESOURCE REQUIREMENTS 49
1. Virtualization Module Program 49
2. VirtualBox 50
3. Graphical Network Simulator-3 51
D. SCALABILITY 52
1. VirtualBox 52
2. Graphical Network Simulator-3 53
E. SUMMARY 54

VI. CONCLUSIONS AND FUTURE WORK 55
A. OVERVIEW 55
B. CONCLUSIONS 55
1. Research Objectives 55
2. Research Questions 56
C. FUTURE WORK 57
1. Software Licensing 57
2. Handheld Device Modeling 57
3. Build Your Own Network Module 58

LIST OF REFERENCES 61

INITIAL DISTRIBUTION LIST 65

LIST OF FIGURES

Figure 1. MAVNATT Conceptual Model [1, p. 58] 6
Figure 2. MAVNATT Tkinter GUI Displaying Network Nodes [1, p. 72] 8
Figure 3. VirtualBox GUI [3] 9
Figure 4. GNS3 with the MAVNATT Conceptual Network Devices 13
Figure 5. MAVNATT Implementation of a WANem Appliance 14
Figure 6. Simplified MAST Architecture [9, p. 27] 15
Figure 7. GraphML Attributes [11] 20
Figure 8. VirtualBox Virtual Host Machine Baseline RAM/CPU Utilization 24
Figure 9. VirtualBox Virtual Host with Server 2012, 2048MB RAM Allocated 25
Figure 10. GNS3 Node and Edge Relationship 26
Figure 11. WANem Appliance GUI 28
Figure 12. VMWare vCenter Converter Concept [14] 30
Figure 13. MAVNATT Virtualization Module Conceptual Network 35
Figure 14. Conceptual Network GraphML Input File pages 1–2 36
Figure 15. Conceptual Network GraphML Input File pages 3–4 37
Figure 16. Conceptual Network GraphML Input File pages 5–6 38
Figure 17. VMP Rendering of Conceptual Network GraphML Input File 39
Figure 18. JSON Excerpt from the Topology.gns3 File 40
Figure 19. WANem Bridge Group Configuration 42
Figure 20. VMP Network Topology within Tkinter from GraphML Input File 46
Figure 21. IP Filtering Functionality Test Network 48
Figure 22. Network Partitioning Through IP Filtering 49
Figure 23. Virtualization Module Conceptual Network, without VMs 51
Figure 24. Complete Virtualization Module Conceptual Network Running within GNS3 52
Figure 25. Enabling VRDE in VirtualBox 53
Figure 26. GNS3 Remote Configuration Settings 54
Figure 27. Android-x86 Running in VirtualBox Hypervisor 58

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CAG	Cyber Academic Group
CLI	command line interface
COTS	commercial off the shelf
CPU	central processing unit
DISA	Defense Information Systems Agency
DNS	domain name system
DOD	Department of Defense
GNS3	Graphical Network Simulator-3
GUI	graphical user interface
GUID	globally unique identifier
HDD	hard disk drive
IOS	Internetwork Operating System
IP	Internet Protocol
IT	information technology
JELA	Joint Enterprise License Agreement
JSON	JavaScript Object Notation
KILSWITCH	Kinetic Integrated Low-cost SoftWare Integrated Tactical Combat Handheld
KMS	Key Management System
LAN	local area network
MAC	media access control
MAST	Malicious Activity Simulation Tool
MAVNATT	Mapping, Awareness, and Virtualization Network Administrator Training Tool
MB	Megabyte
Mbps	Mega-bits per second
MM	malware mimics
MVMC	Microsoft Virtual Machine Converter
NPS	Naval Postgraduate School
OS	operating system

P2V	physical to virtual
PCAS	Persistent Close Air Support
RAM	random access memory
RDP	Remote Desktop Protocol
SATCOM	satellite communications
STIG	Security Technical Implementation Guide
TCP/IP	Transport Control Protocol/Internet Protocol
TDN	Tactical Data Network
USB	universal serial bus
VM	virtual machine
VMDK	virtual machine disk
VMP	Virtualization Module Program
VMIR	Virtualization Module Image Repository
VRDE	Virtual Remote Desktop Extension
WAN	wide area network
XML	EXtensible Markup Language
XSD	XML Schema Definition

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge the love of my life, Mrs. Cheryl Berndt. You are such a graceful and selfless mother, a capable and resourceful homemaker, and the best friend that a guy could ask for. Over the past two years, you expertly managed our growing household almost entirely on your own, allowing me to focus on this demanding program. I love you more than this poorly wrought blurb could possibly express.

I also would like to acknowledge my CSO cohorts who taught me such valuable phrases as “if you wait until the last minute, it only takes a minute,” and “it’s only a lot of reading if you do it”; thank you for the laughs, the support, and the epic bike rides. I learned so much from you all, and will miss each and every one of you.

To Mr. John Gibson and Dr. Alan Shaffer: thank you for your guidance, your long-leash approach to thesis advising, and for the laughs we shared during our meetings. I appreciate the enthusiasm you both had for the project, and all of the insight your guidance offered.

Finally, I would like to thank Mr. Ben McGee of the Redstone Arsenal, without whose technical prowess and patience this project would not have gotten off the ground. You made this thesis an absolute pleasure to work on, and I am glad to have had you in my corner.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In order for Department of Defense (DOD) network administrators to maintain a high level of technical readiness, they must train in an environment that resembles as closely as possible the operational network for which they are responsible. Currently, there is no turnkey solution that allows network administrators complete access to an operational network without introducing some level of risk to the stability of that network. Training in a lab environment eliminates the risk element; however, it provides only a stale representation of a network and does not accurately represent the environment the administrators are training to defend. To help mitigate this network administrator training gap, the Mapping, Awareness, and Virtualization Network Administrator Training Tool (MAVNATT) system was pioneered by Naval Postgraduate School (NPS) student Daniel McBride [1], and remains under development at NPS.

The design intent of MAVNATT is to provide the capability of mapping a network topology, assessing and monitoring the operational status of that network, and creating a virtual copy of the network that is fully partitioned from the operational network. The result provides network administrators with complete flexibility in training exercises and scenarios, while ensuring that the operational network is not impacted by any training evolution. The MAVNATT system consists of three discrete modules for mapping, awareness, and virtualization. This thesis addresses the Virtualization Module of the MAVNATT system, which consists of the following components: a virtualization hypervisor, a network simulator, a wide area network (WAN) emulator, and the virtualization module program that integrates these components. The virtualization module of MAVNATT is responsible for the creation of the virtual environment within which a training scenario may be executed.

A. PROBLEM STATEMENT

The Mapping Module of the MAVNATT system maps and enumerates the operational network environment and then compiles an inventory and configuration details of the nodes within that network. The Module then outputs an open-format graph

file containing *node* elements representing computers and other network devices, *edge* elements representing network links, and attribute data describing the configuration of those elements. Once that open-format graph file is created, the Virtualization Module is responsible to build the virtual instance of the network based on the information provided therein. In the development of the MAVNATT virtualization module, we investigated the following research questions:

1. Primary Question

Is it possible to develop an automated process to virtualize the primary components (router, switches, computers, and links) of an operational network, based upon an open-format graphical representation of the network's architecture?

2. Secondary Questions

To what degree of granularity can the MAVNATT designated hypervisor and network simulator application program interfaces (API) be used to mirror the operational network in a virtual instance?

Can the prototype meet the functionality requirements that were identified within McBride's thesis [1, p. 61], in order to provide a useful training environment?

B. ASSUMPTIONS

The development of the prototype Virtualization Module of MAVNATT assumed the following:

- A basic collection of network node and edge attribute data can be passed from the Mapping and Awareness Modules of MAVNATT to the Virtualization Module in the form of an open-format graph file, which can then be utilized to create a virtual copy of the network.
- Data received as input into the MAVNATT Virtualization Module can provide sufficient detail of the operational network's node and edge attributes to create basic virtual machine (VM), virtual network device, and network link functionality within the virtual copy of the network.
- MAVNATT can parse the attribute data for use as inputs into the VM hypervisor and network simulator components of the Virtualization Module by using each tool's respective API.

- Server and workstation operating systems (OS) included within the operational network shall be created from a gold disk build for each respective OS within the Virtualization Module. Custom OSs, applications, and configurations deviating from the gold disk build will be provided by network administrator intervention following the execution of the MAVNATT virtualization process.

C. OBJECTIVES

The main objective of this work was to provide a prototype of MAVNATT's Virtualization Module. Included in the prototype is the ability to receive an open-format graph file as an input, interpret the file to use the MAVNATT virtualization hypervisor and network simulator to create corresponding computers and network devices, and establish basic network connectivity between those devices.

D. BENEFITS OF STUDY

This work generated a prototype of the MAVNATT system Virtualization Module, which provides network administrators with network awareness, simulation, and training functionality of a tactical network. By developing a system that supports these major areas, we enhance the readiness and overall effectiveness of DOD network administrators by creating training opportunities during all stages of training and operations [1, p. 23].

This work increased the understanding of how a virtualized network can be automatically created from an open-format graphical input. It also explored how the virtual components of that network can be automatically configured through that process to provide a host network for executing the Malicious Activity Simulation Tool (MAST) or other network administrator training tools. The creation of a training environment that both duplicates and remains completely partitioned from the operational network was a key benefit of this research.

E. ORGANIZATION

The rest of the thesis is organized in the following manner:

Chapter II: Background and Previous Work. This chapter outlines the purpose for creating a network administrator training environment that closely resembles a target operational network. It then discusses how the MAVNATT Virtualization Module derives a virtual copy of the operational network based upon the input received from the Mapping Module. Also discussed is the Awareness Module's feed into the Virtualization Module to populate the MAVNATT graphical user interface (GUI) in real time with the status of network links and node reachability.

Chapter III: System Requirements and Design. This chapter discusses the node and link attributes of the operational network that are required to create an accurate virtual instance of the target network, as well as the system requirements necessary to accommodate the MAVNATT conceptual network that this prototype uses as a sample input. We also more thoroughly discuss the components of the Virtualization Module and how they address the MAVNATT system's functional requirements. Finally, this chapter also introduces architectural variations of MAVNATT that may enhance the system's functionality.

Chapter IV: System Implementation. This chapter discusses the tools, applications, APIs, and protocols used to create a virtual network from the given input. It also shows how each of the four sub-components were configured to meet the Virtualization Module's functionality requirements. Additionally, resource requirements for the MAVNATT Virtualization Module prototype are identified according to the scale of the operational network being virtualized.

Chapter V: Virtualization Module Prototype Testing. This chapter demonstrates the virtualization component's effectiveness in the automatic creation of a virtual network. It also provides an assessment of how effectively the prototype met the functional requirements of the MAVNATT Virtualization Module, as well as the thesis objectives.

Chapter VI: Conclusion and Future Work. This chapter discusses the successes and limitations of the MAVNATT Virtualization Module's prototype and identifies functional areas that require further refinement in order to mitigate those limitations.

II. BACKGROUND AND PREVIOUS WORK

A. OVERVIEW

MAVNATT is a system that creates a fully partitioned, stateful, and virtualized instance of an operational network, based on the input of an open-format graph file, in order to create a virtual environment for network administrator training. MAVNATT's functionality is achieved through the integration of three discrete components: a Mapping Module, an Awareness Module, and a Virtualization Module. As this thesis focuses specifically on the Virtualization Module, only a brief description of the remaining modules is included here; a more thorough exploration of the MAVNATT architecture is included in McBride's thesis by the same title [1, p. 58]. Figure 1 depicts the inter-relation between the Mapping, Awareness, and Virtualization modules, as well as the progression from the live, or *operational network*, to the resulting virtual network. The diagram further depicts that the operational network feeds the Mapping Module and the Virtualization Module feeds the virtual network. The interface for the *monitor* function is provided by both the Mapping and Awareness modules, and is displayed within the Virtualization Module GUI; the *train* function is also provided by sub-components of the Virtualization Module.

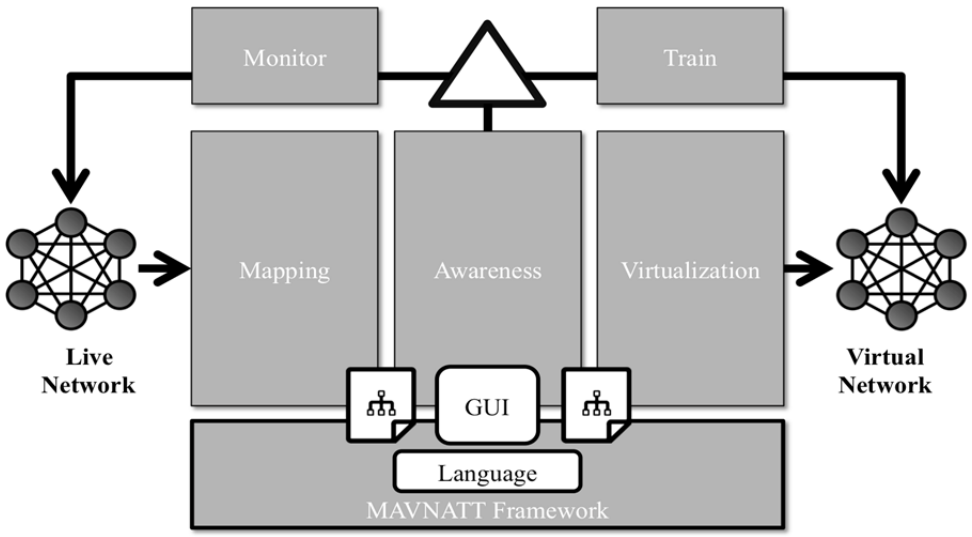

Figure 1. MAVNATT Conceptual Model [1, p. 58]

B. MAPPING MODULE

In the context of network operations, mapping provides the network administrator with an overview of a network by representing its physical layout, interconnectivity of devices, and composition. The MAVNATT Mapping Module provides this capability by automatically enumerating the hosts in an operational network and their associated connectivity [1, p. 33]. The module outputs an open-format graph file with elements representing each host on the operational network, and attributes for each of the hosts. These attributes nominally include: OS, number and type of each network connection, Internet Protocol (IP) and media access control (MAC) address of each connection, next-hop device to determine network topology, as well as other system attributes and states as might be deemed necessary by the system designer [2, p. 58]. After the Mapping Module has populated the open-format graph file, it may receive periodic network status updates from the Awareness Module to reflect changes to the network topology, for instance. Once the graph file is created, it is then presented to the Virtualization Module to be parsed for creation of a virtual copy of the operational network.

C. AWARENESS MODULE

The MAVNATT Awareness Module provides visualization of the operational network topology and fault detection capabilities in order to provide situational

awareness to the network administrator [1, p. 39]. This module employs active monitoring techniques to determine the reachability of each host on the operational network, and reflects topology changes in the network. It periodically updates the MAVNATT Mapping Module [1, p. 59] to provide the network administrator with graphical and alert-based situational awareness of the network.

D. VIRTUALIZATION MODULE

The MAVNATT Virtualization Module bears responsibility for parsing the open-format graph file input, then creating a virtual instance of the operational network based on the attribute data contained therein. To that end, this research focuses on the development and integration of four sub-components within the Virtualization Module: the Virtualization Module Program (VMP), the virtualization hypervisor, the network simulator, and the wide area network (WAN) emulator.

1. MAVNATT Virtualization Module Program

The MAVNATT VMP is a Python-based program that parses the open-format graph input file provided by the Mapping Module to determine the number of nodes, the type of device represented by each node, and each node's next hop. It then uses the APIs for the hypervisor and network simulator to create the virtual network. The VMP utilizes the Python Tkinter GUI, which generates a graphical representation of the virtual network as seen in Figure 2. Additionally, this GUI provides the network administrator with functionality to interface the VMs and virtual network devices on the virtual network.

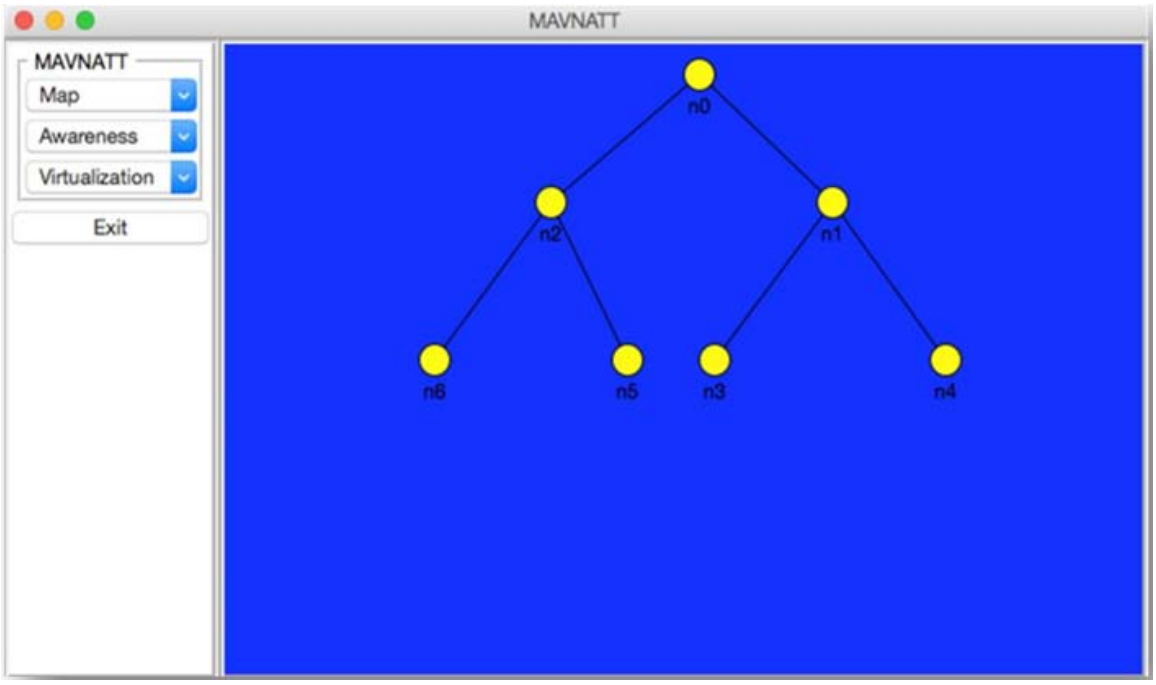

Figure 2. MAVNATT Tkinter GUI Displaying Network Nodes [1, p. 72]

The MAVNATT GUI provides the network administrator with the following capabilities:

- Allows the execution of a new mapping process of the operational network by calling the Mapping Module, which enumerates the devices and their associated connectivity. Upon successful execution of the mapping process, an open-format graph file is populated with the node data gathered from the operational network, and is then ready to be used as an input into the Virtualization Module. Each time the VMP calls the Mapping Module, it pulls the latest updates from the Awareness Module before building a new open-format graph input file.

- Provides the ability to load a previously populated open-format graph file to create a virtualized instance of a network. This feature eliminates the necessity for MAVNATT to have live connectivity to the operational network and provides the network administrator with greater flexibility in conducting training operations in austere environments.

- Creates a graphical representation of the operational and virtual networks' topologies, including the status of the network links and reachability of each node within the networks. The input received from the MAVNATT Awareness Module provides the link and node status that is displayed within the Virtualization Module GUI.

2. MAVNATT Virtualization Hypervisor

McBride's thesis established an architecture and framework for MAVNATT, including the evaluation of various virtualization hypervisors and network simulators for use in the Virtualization Module. This evaluation resulted in the selection of the VirtualBox® hypervisor due to its wide range of OS support, ready availability of APIs supporting integration with other applications, open source availability, and exportability of virtual device images to other virtualization products [1, p. 46]. Figure 3 shows the VirtualBox user interface in a typical implementation.

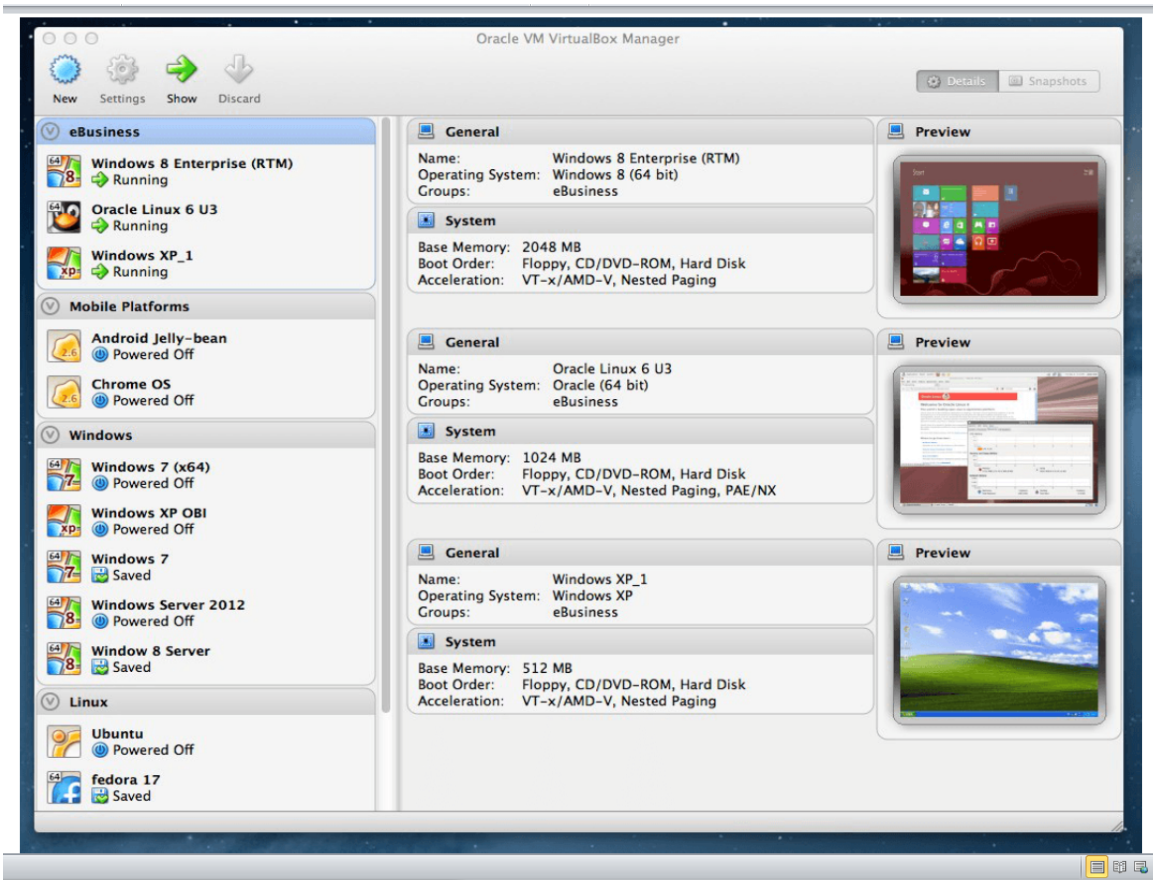

Figure 3. VirtualBox GUI [3]

The VirtualBox hypervisor officially supports many commonly used OSs, however it was developed to provide a generic x86 virtual environment and, as such, may run any x86 compatible OS. Officially supported guest OSs include:

- Windows 2000/XP, Server 2012/2008/2003, Vista, Windows 7/8, Windows NT 4.0, Windows 95/98/ME, Windows 3.x, DOS
- Mac OS X, through version 10.11
- Linux 3.x/2.6/2.4
- Solaris 11/10
- FreeBSD, OpenBSD
- OS/2 Warp 4.5

The MAVNATT virtualization hypervisor provides the capability of creating a VM corresponding to each computer node in the operational network, as discovered by the MAVNATT Mapping module. Using the open-format graph file received from the Mapping Module, the hypervisor analyzes each network node for attributes such as OS version, information about system resource allocation including: Hard Disk Drive (HDD) size, random access memory (RAM) allocation, and number of processors. It also takes into account network interface settings such as: IP address, subnet mask, default gateway, domain name system (DNS), and MAC address.

The Virtualization Module uses the MAVNATT virtualization API to create VMs corresponding to each OS version within the operational network. These VMs are cloned by the VMP, then loaded into the hypervisor as virtual machine disk (VMDK) files from a preconfigured gold disk of the OS that resides within the MAVNATT Virtualization Module Image Repository (VMIR). Since MAVNATT is ultimately intended for use on the Marine Corps Tactical Data Network (TDN), the standard OS versions and builds typically used on that network can be included in the VMIR. In an environment with a well-controlled configuration management process, such as the TDN, achieving OS parity between the operational network and the virtual network is a realistic goal for MAVNATT to achieve. Each image included in the VMIR has a corresponding VMDK file that is used by the virtualization hypervisor to create the VMs with which network administrators will interface during training scenarios. Once a VM is created, the Virtualization Module uses the hypervisor's API to overlay attributes specific to each

entity based on operational network data received in the open-format graph file, as captured by the MAVNATT Mapping Module.

Upon creation of the virtual network, the MAVNATT Virtualization Module provides network administrators the flexibility with which to interface each VM in order to affect additional customization steps. The MAVNATT Virtualization Module uses the hypervisor's and network simulator's APIs to leverage the native console functionality of both tools. This functionality allows the network administrator to modify the settings of all VMs and network devices through the MAVNATT GUI, and to load any additional software that may be required in order to duplicate the operational network as closely as possible.

3. MAVNATT Network Simulator

A MAVNATT network simulator provides network connectivity between VMs within the MAVNATT Virtualization Module, as well as the ability to create the routers and switches discovered by the Mapping Module and captured within the open-format graph file. The simulator interfaces directly with both the virtualization hypervisor and the MAVNATT Virtualization Module program. MAVNATT uses the network simulator's API to assign IP addresses to each live interface on the modeled devices according to the associated data contained within the open-format graph file. In this prototype, the Virtualization Module uses an EXtensible Markup Language (XML) based GraphML input file as its input, which includes the IP addresses of each router's interfaces. The Virtualization Module uses the network simulator's API to assign the IP address to the virtual router's respective interfaces. Each IP network to which the virtual router is connected is advertised within the routing protocol configuration of the router. The routing protocol then dynamically builds the routing table and establishes layer 3 connectivity between the virtual network nodes. In MAVNATT's development end state, the Mapping Module will provide the topology information to the Virtualization Module.

McBride's thesis selected the open-source Graphical Network Simulator–3 (GNS3), created primarily to provide a training platform for Cisco and Juniper network devices, as the network simulator used by MAVNATT [4]. GNS3 is written in the same

Python programming language as MAVNATT and has extensive API availability, which makes it a strong choice for use in the MAVNATT Virtualization Module. Additionally, GNS3 provides the ability to emulate supported Cisco devices running a Cisco internetwork operating system (IOS) image that has been hardened to achieve Defense Information Systems Agency (DISA) Security Technical Implementation Guidelines (STIG) compliance [5]. GNS3 provides the flexibility to interface with both physical and virtual devices, both within and outside of the GNS3 environment, which will allow MAVNATT to incorporate network devices that are not natively supported by the GNS3 platform.

The MAVNATT Mapping Module can transfer a .cfg configuration file from a network device on the operational network directly to GNS3, where the file can then be loaded by the corresponding virtual network device upon system boot [4, p. 33]; this capability enables configuration parity of network devices between the operational and virtual networks. GNS3 also integrates natively with VirtualBox, which eliminates the need for further complexity within the MAVNATT Virtualization Module design. Figure 4 depicts the GNS3 interface loaded with a GraphML input file representing the network devices from the MAVNATT conceptual network.

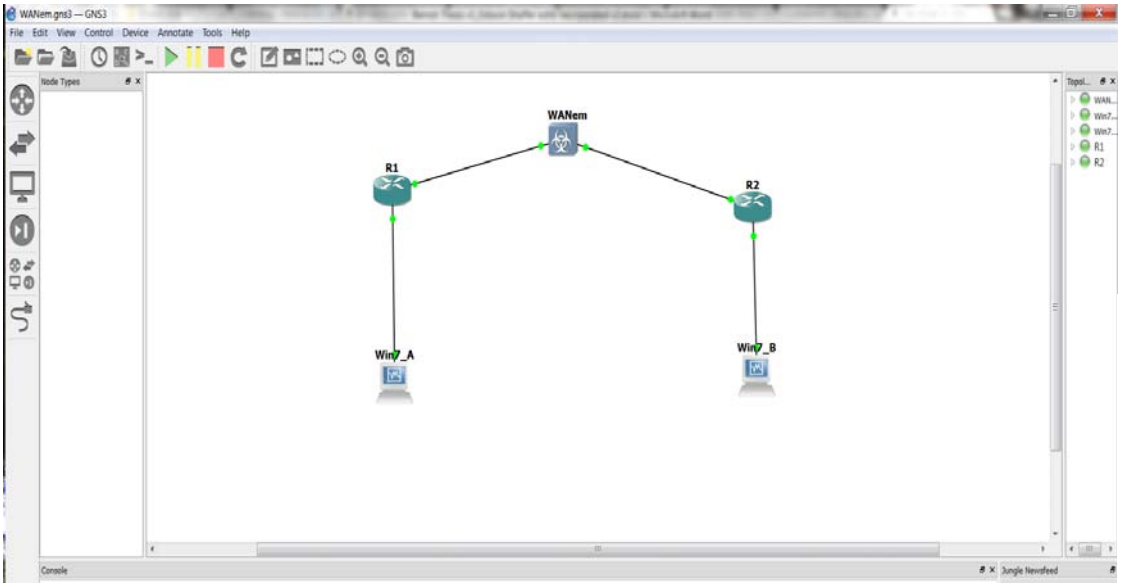

Figure 4. GNS3 with the MAVNATT Conceptual Network Devices

4. MAVNATT WAN Emulator

The Marine Corps' TDN typically includes satellite communication (SATCOM) links, as well as other sub-megabit per second (Mbps) WAN links in its topology [6]. As such, there is a need to emulate these high-delay, low-bandwidth circuits within the MAVNATT virtual network in order to accurately mimic the functionality of the operational network. To this end, the MAVNATT Virtualization Module employs a WAN emulator to integrate with the network simulation component and create these specialized WAN link characteristics as dictated by the operational network's topology.

In the MAVNATT implementation of WANem, the emulator is bridged between two virtual routers, as depicted in Figure 5, to provide the MAVNATT Virtualization Module the ability to mimic WAN characteristics found in the operational network. For example, each network hop from a terrestrial source to a geosynchronous satellite, then back to another terrestrial destination, introduces a delay of up to 500 milliseconds into the circuit by the time a packet makes a round trip [7]. This latency can significantly impact the performance of a network, and without factoring this characteristic of the operational network into the virtual environment, a performance parity between the networks cannot be achieved.

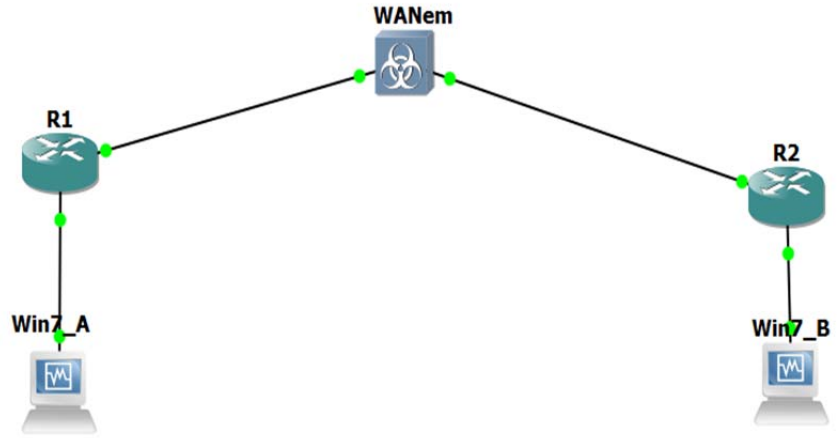

Figure 5. MAVNATT Implementation of a WANem Appliance

5. Integration with the MAST Training Tool

Central to MAVNATT's functionality goals is the ability to allow network administrators to conduct scenario-based cyber training in an environment that is completely partitioned from the operational network [1, p. 39]. The malicious activity simulation tool (MAST) was designed to provide such training through the use of malware mimics (MM), as described in the NPS thesis by Nathaniel Hayes [8]:

> MAST, a software program under development at NPS, mimics the behavior and impact of network-based malware in an effort to train the administrators of operational DOD networks both to respond to the threats such materials present to their networks and to assess their competence in recognizing and responding to such threats.[p. i]

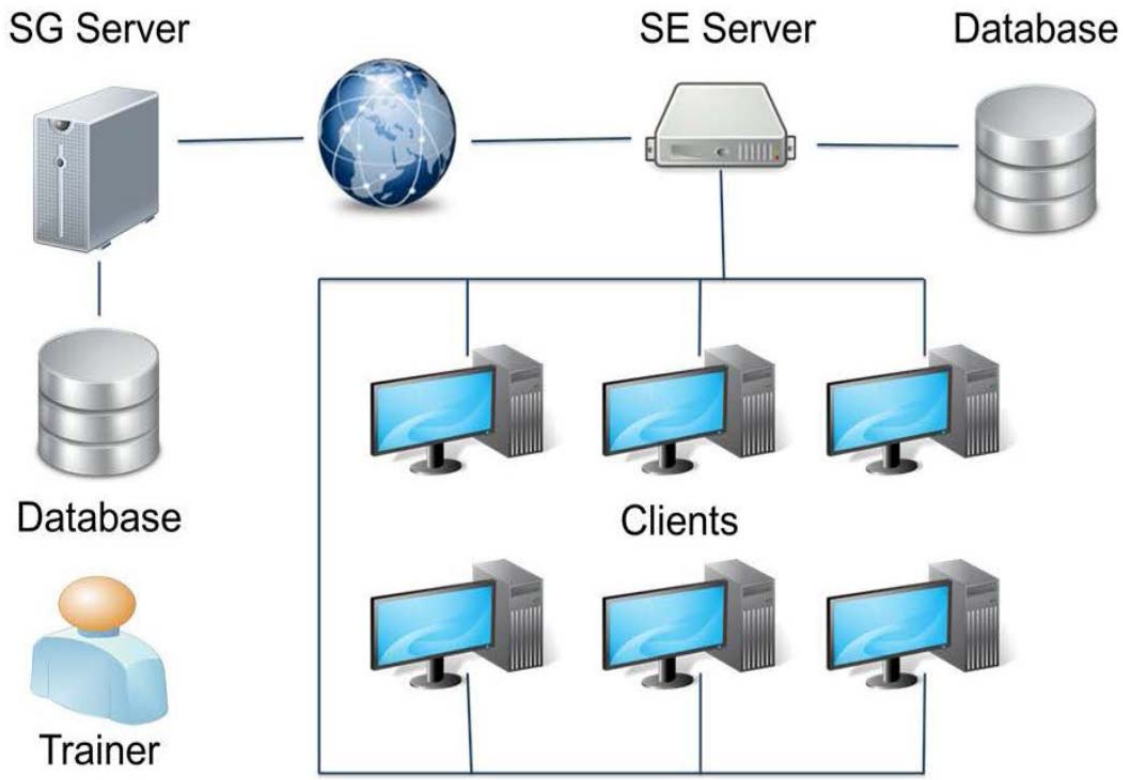

Figure 6. Simplified MAST Architecture [9, p. 27]

As depicted in Figure 6, the MAST architecture consists of a Scenario Generation (SG) server, a Scenario Execution (SE) server, and multiple MAST clients; each of these roles is an individual Java program designed to run on a Microsoft Windows environment, though not limited to that environment. The SG server maintains a database of scenario modules, which are developed by trainers with experience in ethical hacking or Red Team disciplines. Each module mimics a specific behavior, such as a network scan or a ping; multiple modules are assembled together by the trainer to create a training scenario file. The scenarios are then deployed to the SE server, which is typically located remotely from the SG server at the site of the MAST training. The SE server pushes the scenarios to the appropriate MAST clients to execute each scenario. The SE server also maintains a local scenario database and can operate independently from the SG server, following the initial deployment of scenario files to the SE server or creation of such files locally.

A related NPS thesis by Ray Longoria discussed an example of a typical MM within a MAST scenario:

> In this scenario, a pop-up window appears on the user's desktop. The window is a simple image that performs no action other than recording the user's response. The pop-up window asks the user if they would like to execute or download a specific file. The user's actions are recorded in the SE Server's database.
>
> The objectives of this scenario are to see how the users respond to the download question and if any users report the events to a system or network administrator. Such events may be characteristic of a phishing attack. The results of the training can let a unit know where to focus future training resources. [9, p. 29]

VMDK files of the SG, SE, and MAST client machines are included in the Virtualization Module Image Repository (VMIR), which allows the MAVNATT Virtualization Module to create a virtual MAST environment. The Virtualization Module uses the MAVNATT hypervisor's API to provide the network administrator with an interactive console session with the MAST servers and clients. Once a GraphML input file representing a MAST environment has been loaded into MAVNATT, the MAST administration and scenario functionality are identical to the physical installation of MAST.

E. SUMMARY

This chapter provided a general overview of MAVNATT and its three modules, and their relationship to this research. Chapter III will discuss the functional requirements of the Virtualization Module, and how the prototype system design aims to meet those requirements. Additionally, it thoroughly discusses the Virtualization Module's components.

III. SYSTEM REQUIREMENTS AND DESIGN

A. OVERVIEW

The previous chapter provided a basic overview of the MAVNATT system, as well as an introduction to the components of its Virtualization Module. In this chapter we more thoroughly discuss those components and how they address the MAVNATT system's design requirements. The Virtualization Module functional requirements defined in McBride's thesis directly influenced the framework of this prototype design and formed the basis for the testing described in Chapter V.

B. VIRTUALIZATION MODULE PROTOTYPE REQUIREMENTS

As outlined in McBride's thesis, the main objective of the Virtualization Module is to integrate MAVNATT with a hypervisor that can support the virtualization of a network topology that is logically partitioned from the operational network. The MAVNATT Virtualization Module prototype described here was developed to meet the following functional requirements identified in the thesis [1, p. 60]:

Requirement: The module must communicate with the Awareness Module. Specifically, it should receive a network topology to virtualize and return a set of connectors to those virtual devices.

Solution: We discussed in the previous chapter that the MAVNATT Awareness Module was not yet developed at the time this research was conducted, therefore a functional demonstration of the Awareness-Virtualization Module interface cannot be provided at this time. In lieu of actual inputs from either of the Mapping or Awareness Modules, this prototype makes use of a static GraphML file that represents a conceptual network. The prototype is able to read the *node* and *edge* object data within the GraphML input file, and then control the hypervisor and network simulator through their respective APIs to build the corresponding virtual devices. The prototype is also able to configure those devices according to the attribute data contained within the file for each *node* and *edge*. Both VirtualBox and GNS3 make their entire feature sets available to developers through their respective API's [2] [4]; as such, any device setting or state can be passed from the

VMP to the hypervisor or network simulator as a GraphML *node* or *edge* attribute. This capability provides excellent flexibility for future integration with the Awareness and Mapping Modules.

Requirement: The module must ensure the virtual devices are unable to interfere with the operational network.

Solution: The MAVNATT Virtualization Module creates a virtual copy of the operational network. The resulting virtual copy is completely partitioned from the operational network, with one exception: the machine hosting the Virtualization Module is configured with an IP filtering policy [10] that restricts all outbound traffic from the virtual network. A single *permit* rule enables the Virtualization Module to communicate with the Mapping and Awareness Modules; these modules are installed on a host within the operational network. This IP filtering policy enables the communication between the virtual and operational networks that is necessary for MAVNATT functionality, while eliminating the threat of the module interfering with the operational environment.

Requirement: The module must infer a virtual device whenever a full specification is not available.

Solution: A key feature of the prototype's design is the Virtualization Module image repository (VMIR), a directory of VMDK and IOS files used to create virtual machines and virtual network devices within the Virtualization Module's environment. The virtual images are baseline copies of OS installations. The VMDK files are loaded into VirtualBox to create VMs. The IOS files are loaded into GNS3 to create virtual network devices. The VMP then applies the node attributes to the VM or virtual network device through the APIs of the hypervisor and network simulator based upon the attribute data contained within the device's corresponding GraphML node. If a particular node contains no data for an attribute, no action is taken to configure that attribute in the VM or virtual network device. If additional configuration is required for the VM or network device, the network administrator has the ability to establish a console session with the device, through the VMP GUI, to apply additional settings manually in the absence of an automatic process.

C. SYSTEM DESIGN

The previous chapter discusses the general design of the MAVNATT Virtualization Module and provides a general overview of how the Virtualization Module Program (VMP), the virtualization hypervisor, the network simulator, and the WAN emulator all integrate to create a virtual network from a GraphML input. This chapter discusses in greater depth how the Virtualization Module leverages each of those four sub-components to meet its functional requirements. Each sub-component of the Virtualization Module, along with the static GraphML input file used as an input to the system, is thoroughly discussed in this chapter.

1. GraphML Input File

When the MAVNATT system reaches its development end-state, the Virtualization Module will have the capability of manually executing a mapping process from the Mapping Module through the VMP GUI. This process will be used as an input to build the initial virtual copy of the network, as well as to synchronize the virtual network with the latest reachability data and topology deltas from the operational network as provided to the Mapping Module by the Awareness Module. Since this thesis was conducted concurrently with the development of the Mapping Module, we developed a static GraphML file, populated with the elements and attributes of a conceptual network, to serve as an input to the system.

The VMP uses several elements of the GraphML input file to store operational network data received from Mapping and Awareness Modules. A *node object* represents each computer and network device and an edge object represents each network link. Each of these *node* and *edge* objects is assigned an ID, a descriptive value, and an X & Y coordinate that determines the object's position on the network diagram. In addition, the object can be configured with various attributes to describe characteristics that can be stored as Boolean, int, long, float, double, or string data types [11]. An *edge* object typically possesses *source* and *target* attributes to establish a connection between two *nodes*; this feature of the GraphML language is used by the VMP to create the network topology. These attributes are stored for each object as data *keys* and can be set arbitrarily

by the system designer. Figure 7 shows an example of a simple GraphML file with six defined *nodes*—some with data *keys* and others without—and associated *edges* between the *nodes*.

Example of a GraphML Document with GraphML-Attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="d0" for="node" attr.name="color" attr.type="string">
    <default>yellow</default>
  </key>
  <key id="d1" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```

Figure 7. GraphML Attributes [11]

The GraphML schema can be extended to add any data key that is needed in order to further define each node or edge. The key values are defined at the top of the GraphML object and can be contracted or added to as needed. Specifically, each node is an object and has an associated mapped graphical icon associated with it for the GUI. As the GraphML input file is read into MAVNATT at run time, NetworkX combines with the python Tkinter library to draw the icons for each node, then connects lines between specific nodes based upon the edge objects in the GraphML input file.

Each *node* must be defined as a type of object: computer, router, or switch. Each key has an attribute name defined as *attr.name* and an attribute type defined as *attr.type*.

An ID is included for each attribute, which corresponds to a *data key* entry. This pairing associates a data value with a specific attribute for either a *node* or an *edge*, as specified by the *for* value. The following example shows a source IP address stored as a string value in an edge attribute. It also shows that each key can be assigned a *for* value of either *edge* or *node*:

<key attr.name="sourceIP" attr.type="string" for="edge" id="d1" />

<data key="d1">10.0.0.1</data>

Each *edge* can be constructed as shown in this example:

<edge source="switch.syscon" target=" Windows7_A ">

<data key="d6">192.168.1.2</data>

<data key="d7">eth0</data>

<data key="d8">192.168.1.21</data>

<data key="d9">eth1</data>

</edge>

This *edge* shows a network link from a switch at *192.168.1.2* to a machine at *192.168.1.21*.

Each *node* is constructed as shown in this example:

<node id="Windows7_A">

<data key="d0">200</data>

<data key="d1">250</data>

<data key="d2">computer</data>

<data key="d3">Windows7</data>

<data key="d10">C:\MAVNATT\VMIR\Windows7_A.vmdk</data>

</node>

This *node* shows a Windows 7 computer, as well as the VMIR directory and the specific VMDK file the Virtualization Module will use to clone it.

2. Virtualization Module Program

The MAVNATT Virtualization Module Program (VMP) is a Python program with a Tkinter GUI. Within the GraphML input file, each network device is represented as a *node* and each network link is represented as an *edge.* The VMP parses the GraphML input file in order to categorize each *nod*e and *edge* represented within the file, along with its attributes. It then uses the API of either the hypervisor or the network simulator to create a virtual machine or a virtual network device, respectively. Once a GraphML input file has been populated with the network *nodes*, *edges*, and associated attributes, it needs to be processed in order to create the virtual network. The VMP reads the GraphML input file to first draw the *nodes* and *edges* in the MAVNATT GUI. This is accomplished using the NetworkX library within the Python/Tkinter library for the GUI. A validation routine runs to ensure that the minimum information needed in order to virtualize is provided in the GraphML input file; at a minimum, each *node* must include the device type attribute. If no value is contained for this attribute, no action is taken to virtualize the device and an exception is thrown; the administrator is then prompted to manually enter the device type, which results in the creation of a generic device in the virtual network. The administrator can then interface directly with the device to perform additional customization following the virtualization process.

3. Virtualization Hypervisor

The MAVNATT Virtualization Module employs the VirtualBox hypervisor to create the VMs within the virtual network. VirtualBox provides an extensive software development kit (SDK), which allows its *Main API* to control the entire VirtualBox feature set [2, p. 21]. For this proof of concept, the MAVNATT VMP uses the VirtualBox API functionality to perform the following actions as it creates virtual machines from the GraphML input file:

- Reads the OS type and then creates a VM clone from the corresponding VMDK file in the Virtualization Module Image Repository (VMIR)

- Reads the hostname, IP address, MAC address, and IP address of the next-hop device or devices to determine the network topology
- Applies these attributes to the VM, then moves to the next VM or network device in the GraphML input file

In the MAVNATT development end state, the VMP will have the capability of passing any attributes supported by the VirtualBox platform [2, p. 52] from the GraphML input file provided by the Mapping and Awareness Modules to the VM clone. More specifically, if a setting or configuration of a computer residing on the operational network can be captured by the Mapping and Awareness Modules it can be reproduced within the VirtualBox hosted VM in the virtual network. This provides great potential for achieving near-parity between the operational network and its virtual copy.

An inherent characteristic of a hypervisor is that the RAM requirements of the individual VMs place a corresponding demand upon their host machine. Figures 8 and 9 illustrates this characteristic the first figure shows a test host machine's baseline RAM utilization level at 2.2GB. Upon starting a single Windows 7 VM with 2GB of allocated RAM, the RAM utilization of the host machine correspondingly increases to provide the VM with the allocated resources.

While this memory usage did not adversely impact the implementation of our prototype system, it will require additional consideration as MAVNATT is used to map large-scale operational networks.

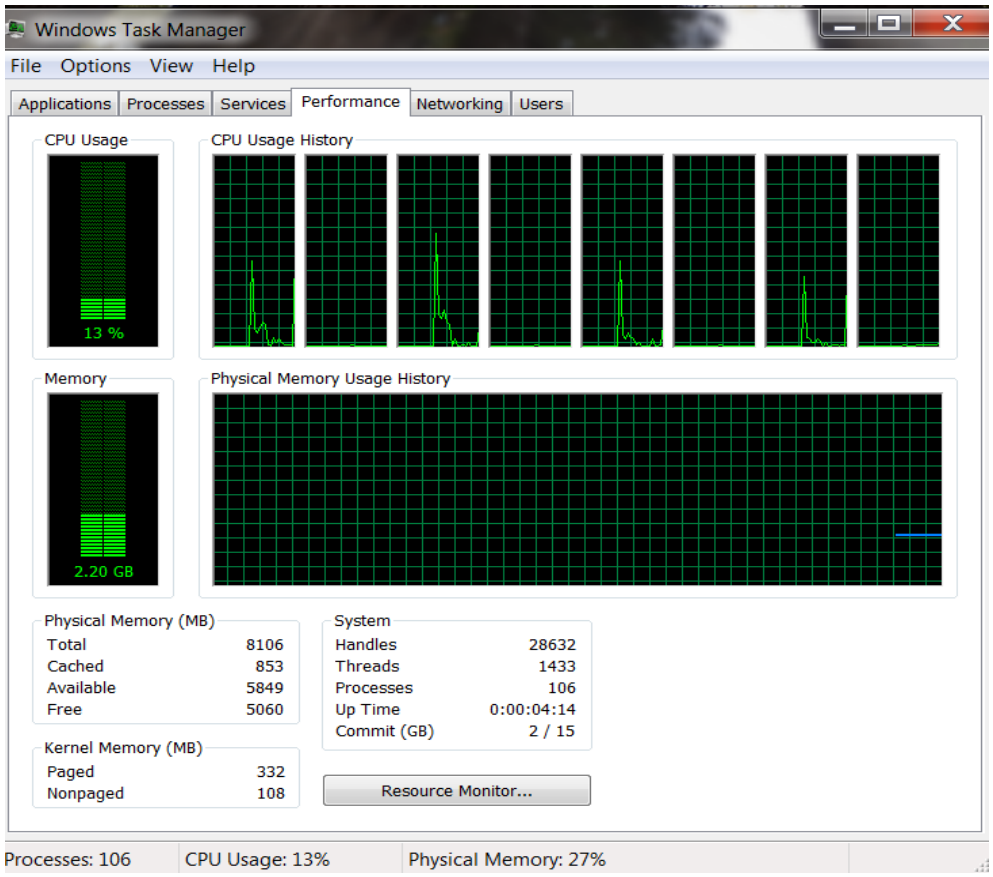

Figure 8. VirtualBox Virtual Host Machine Baseline RAM/CPU Utilization

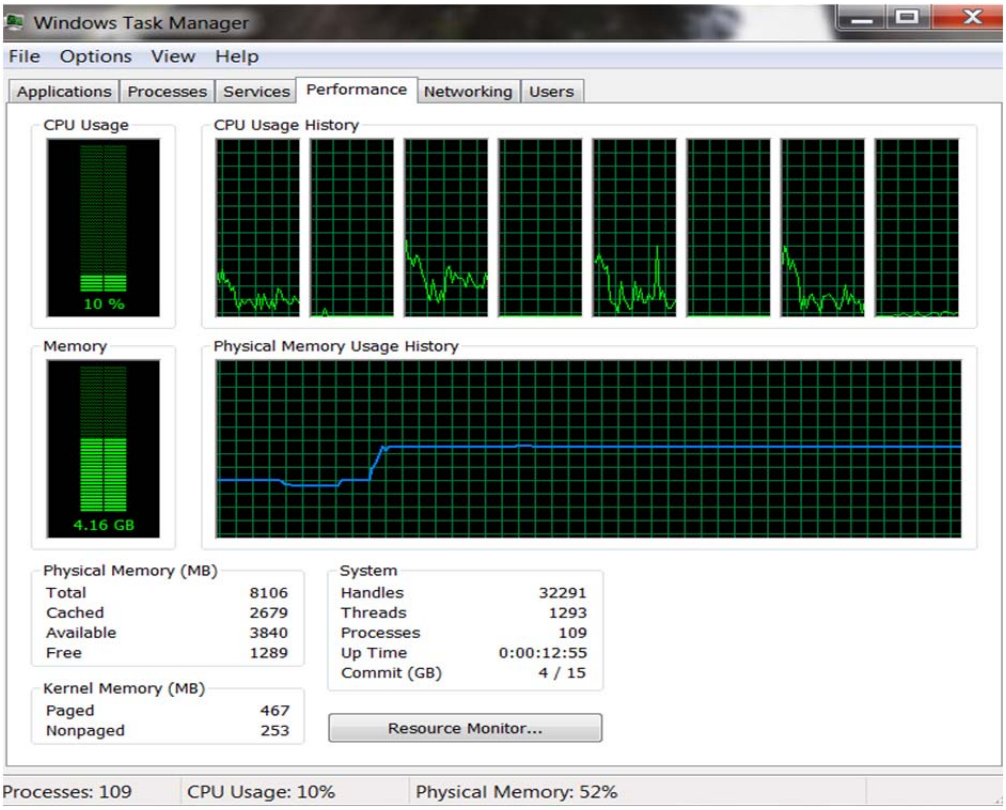

Figure 9. VirtualBox Virtual Host with Server 2012, 2048MB RAM Allocated

4. Network Simulator

The MAVNATT Virtualization Module uses the GNS3 network simulator to provide virtual TCP/IP connectivity between VMs in the virtual network. After the GraphML input file is validated by the VMP the user can click the *virtualize* button. This *click event* takes the GraphML data and builds the necessary supporting files for the GNS3 project to be created. These GNS3 files include the new project directory, the *project.gns* JavaScript Object Notation (JSON) file, the subfolder of *dynamips* for network devices, and a *virtualmachines* subdirectory for the VMs created in the project. Within the *dynamips* directory, there is a subdirectory called *config*, where the Cisco *IOS.cfg* files are generated based on network device parameters passed by the GraphML input file. GNS3 uses an internal program to clone network devices with the CLI. After the cloned image is created, the IOS.cfg file is added for specific settings for each

network device. After creating the underlying project files, the VMP launches GNS3 from the CLI and passes the project.gns file to the program at run time.

In the process of constructing the nodes and edges from the GraphML input file, GNS3 builds a JSON file to represent each device within the GNS3 project. Figure 10 shows how GNS3 uses this file to represent the network edges for the GraphML input file, with topological relationships between nodes in the prototype network.

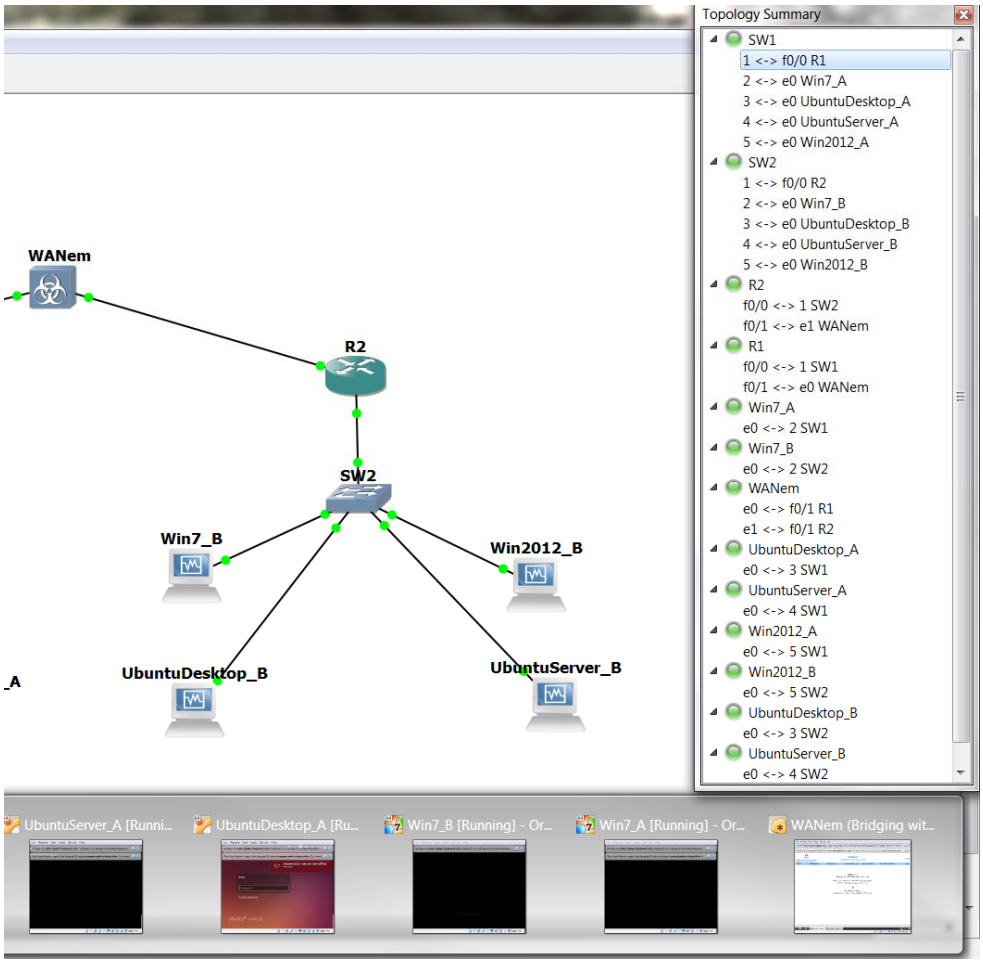

Figure 10. GNS3 Node and Edge Relationship

5. Wide Area Network Emulator

WANem is an application that runs on a Knoppix Linux distribution and is configured through a web GUI. The WANem installation is initially loaded into the VirtualBox hypervisor as a bootable International Organization for Standardization (ISO)

image file, then exported to the VMIR as a VMDK file. This portability allows a number of WANem appliances to be added to a virtual network, as dictated by the characteristics of the operational network. In the MAVNATT implementation of WANem, the network administrator must manually configure the desired WAN characteristics for each link due to the unavailability of a suitable API. Characteristics that are commonly found in the tactical network environment can be saved as WANem rule sets, and then loaded for rapid deployment of an appliance to emulate a particular type of link.

The MAVNATT Virtualization Module incorporates the WANem appliance into its VMIR in order to provide the capability of emulating various WAN characteristics that may exist within the operational network. These characteristics include network delay, packet loss, packet corruption, disconnections, packet reordering, jitter, etc. [12]. Figure 11 shows the wide range of configurable parameters provided by WANem, which allows us to duplicate virtually any combination of circuit characteristics in the Virtualization Module.

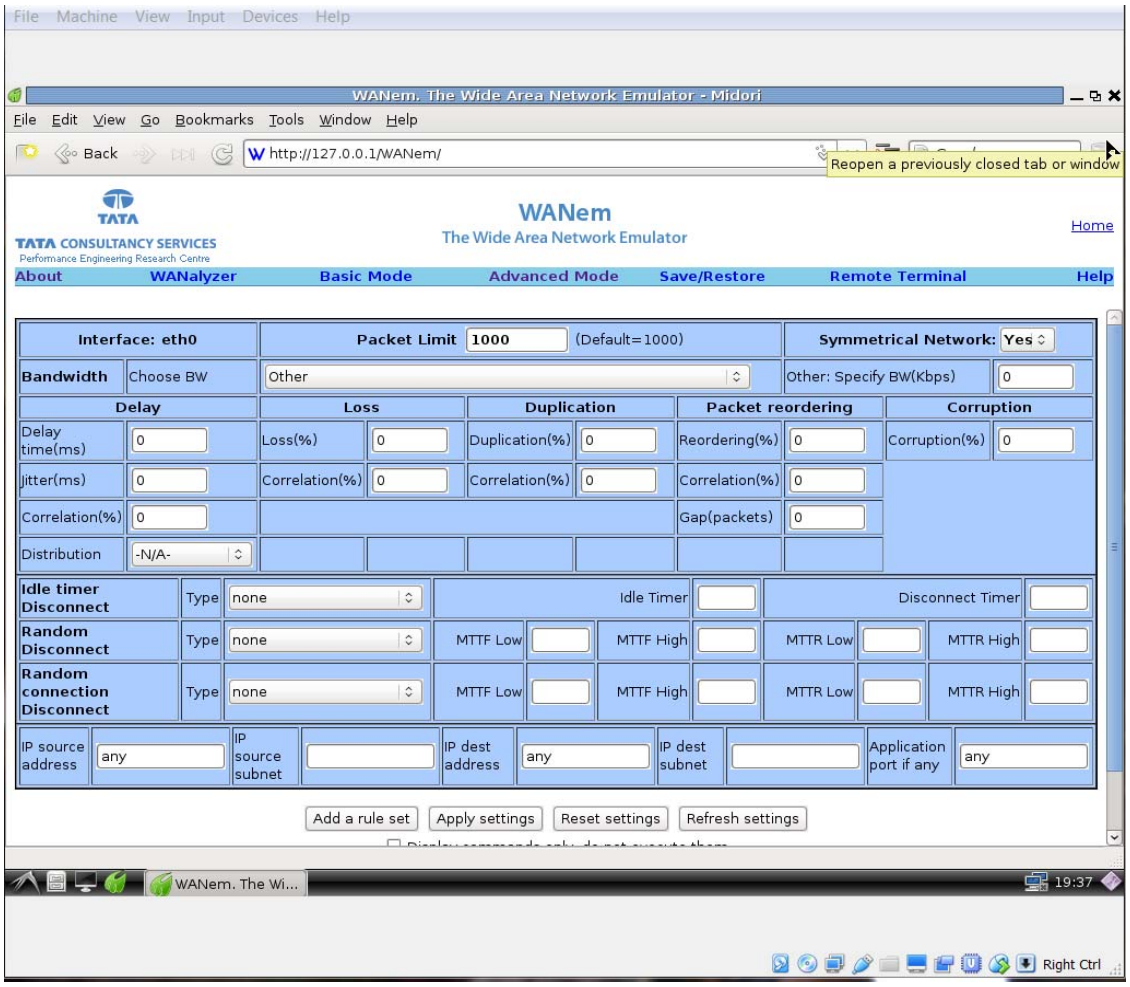

Figure 11. WANem Appliance GUI

6. Physical-to-Virtual Process

a. P2V Overview

Although not incorporated into this prototype's development, another design consideration for the Virtualization Model is to provide the ability to perform a physical-to-virtual (P2V) conversion of the server and workstation machines within the operational network. The P2V process is common in the migration of a physical environment to a virtual environment, and is a generic term for capturing the complete data and state of a computer, then creating a file from that data which can be imported into a virtualization hypervisor as a virtual machine [13]. In the P2V process, all of the

physical drivers used by the source machine are replaced with virtual drivers, which preserves the original functionality of the system.

b. P2V Benefits

Although P2V conversion deviates from the original design intent of the MAVNATT Virtualization Module, which is the use of an open-format graph file as a model of an operational network, there may be several benefits to incorporating P2V into the MAVNATT architecture. P2V products such as the VMWare vCenter Converter can output a virtual machine file which results in a virtual instance of a server or workstation machine that is functionally identical to the physical version [14]. Additionally, Microsoft offers the native P2V tools Microsoft Virtual Machine Converter (MVMC) and Disk2vhd for use in converting Microsoft server and workstation OSs, respectively [15].

The primary benefit provided by P2V tools is the functional parity between the operational and virtual networks that result from the P2V process. Whereas the current VMIR-based VM cloning process utilized by MAVNATT requires network administrator intervention to install software applications and customizations that deviate from the gold disk builds, the P2V process copies each physical machine from the operational network, then creates an exact copy in the virtual network. Figure 12 provides a graphical depiction of the VMWare vCenter Converter, and the general P2V concept.

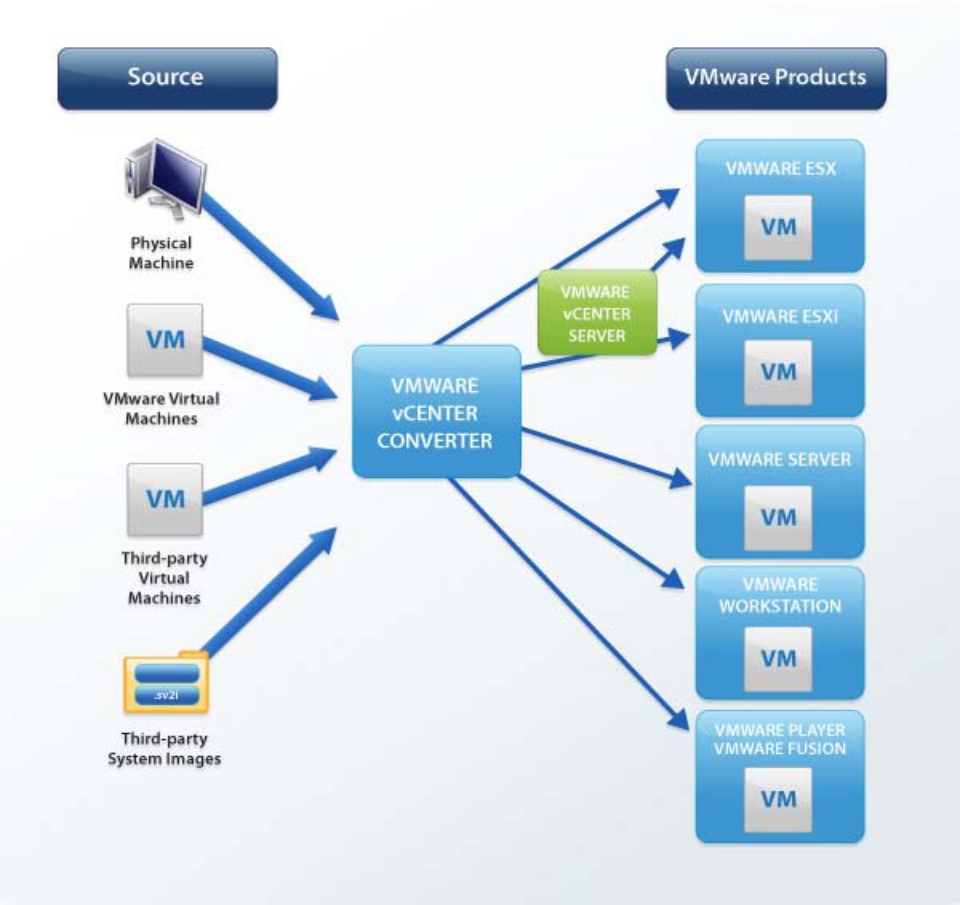

Figure 12. VMWare vCenter Converter Concept [14]

a. P2V Drawbacks

Although the P2V process provides an exact copy of the physical environment within the virtual network, the main drawback is that the process can take an hour or multiple hours for each device [16]. MAVNATT is intended to be used in tactical environments, and as such must be a lightweight solution, meaning that the P2V process would not be feasible as a deployable design. In a more static training environment, the P2V process could provide functionality enhancements over the process provided by the standard MAVNATT architecture. Nonetheless, if used by the supported organization well in advance to field activities a significant database of the organization's systems could be established as part of the library of virtual machines available for instantiation upon deployment through the Virtualization Module Image Repository.

D. SUMMARY

This chapter discussed the MAVNATT Virtualization Module functional requirements, as identified by McBride's thesis, and the prototype system design for meeting those requirements. Additionally, we described the module's components and how each of them is integrated within the system's design. The next chapter provides a thorough description of how the prototype system implements each of the components to achieve the required functionality.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SYSTEM IMPLEMENTATION

A. OVERVIEW

The previous chapter discussed the design of the MAVNATT Virtualization Module, and provided a detailed description of how the Virtualization Module Program (VMP), the virtualization hypervisor, the network simulator, and the WAN emulator were integrated to derive a virtual network from a GraphML file. This chapter discusses in greater depth how each of those four sub-components were configured to meet the Virtualization Module's functionality requirements.

B. COMPONENT IMPLEMENTATION

1. Prototype Design

The Virtualization Module prototype was developed, implemented, and tested in the following environment:

- OS: Microsoft Windows 10
- Host machine: Intel Core i7-2820QM processor @2.4GHz, 64-bit, x86 architecture, 32GB installed memory (RAM)
- Graphical file format: GraphML, version 1.0
- Programming language: Python, version 3.5.0 with NetworkX software library, version 1.10
- GUI: Tkinter, version 8.6.4
- Integrated development environment (IDE): Python IDLE, version 3.5.0
- Virtualization hypervisor: Oracle VirtualBox, version 5.0.10 r104061
- Network simulator: GNS3, version 1.3.11
- WAN emulator: WANem, version 3.0 on Knoppix Linux 6.7.1

2. Virtualization Module Conceptual Network

The Mapping and Awareness Modules of MAVNATT are undergoing development concurrently with the Virtualization Module. As a result, this prototype makes use of a static GraphML file representing a conceptual network. The Virtualization Module conceptual network for this prototype is comprised of three separate IP networks: two LANs, and one WAN. Each of the LANs includes a router; the two routers are connected to each other by a WAN link to simulate a geographical separation between the networks, and to provide an opportunity to inject WAN characteristics into the prototype. The LANs are each populated by the following devices: an Ethernet switch, a Windows Server 2012 server, a Windows 7 workstation, an Ubuntu Linux server, and an Ubuntu Linux desktop, as depicted in Figure 13. This conceptual network is of sufficient scale to satisfy the MAVNATT Virtualization Module functional requirements at this point in the project's development.

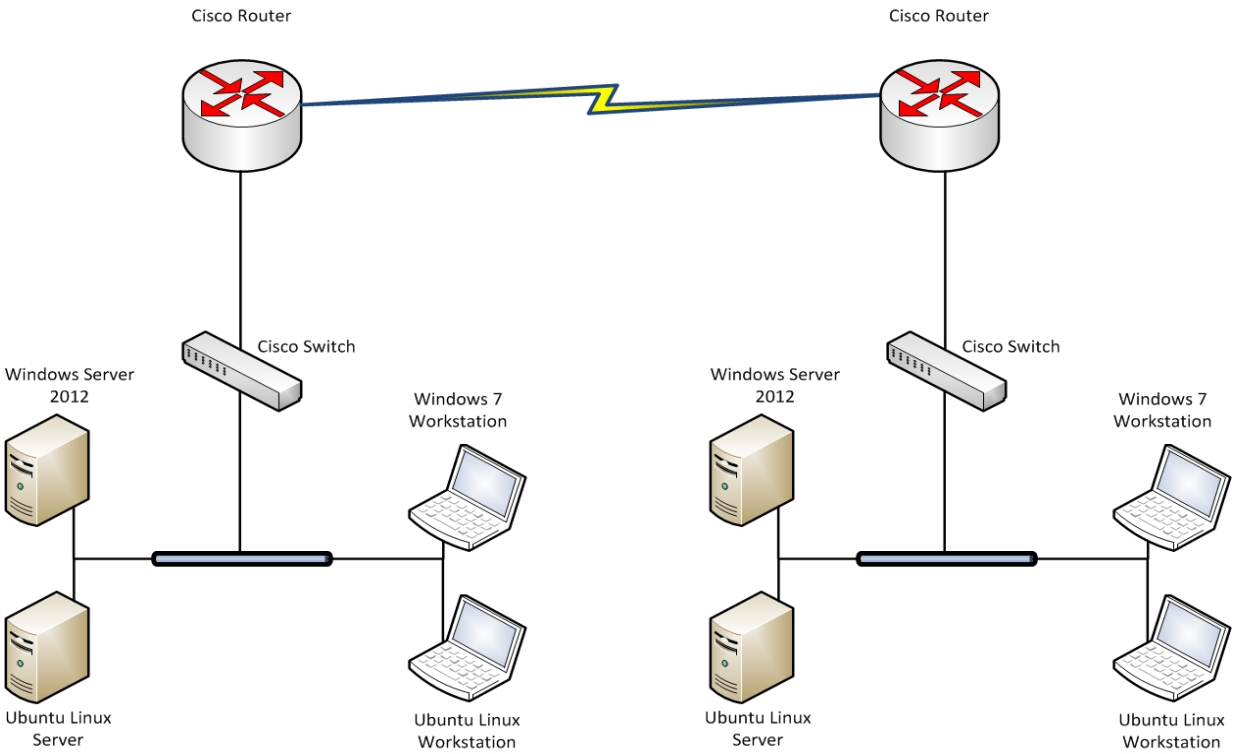

Figure 13. MAVNATT Virtualization Module Conceptual Network

3. GraphML Input File

In the design phase of the Virtualization Module, we developed the conceptual network in order to demonstrate the prototype's ability to meet the MAVNATT functional requirements. In the implementation phase, the first step was to manually generate a GraphML input file to represent the nodes, edges, and attributes of the conceptual network to simulate input from the MAVNATT Mapping Module. Figures 14, 15, and 16 include the GraphML input file code that represent the conceptual network. Each of the node objects is assigned basic attributes including OS type and IP address, and each of the edge objects is assigned connection point attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
          http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
 <key attr.name="oslocation" attr.type="string" for="node" id="d10" />
 <key attr.name="sourceInterface" attr.type="string" for="edge" id="d9" />
 <key attr.name="destinationIP" attr.type="string" for="edge" id="d8" />
 <key attr.name="destinationInterface" attr.type="string" for="edge" id="d7" />
 <key attr.name="sourceIP" attr.type="string" for="edge" id="d6" />
 <key attr.name="x" attr.type="int" for="node" id="d5" />
 <key attr.name="y" attr.type="int" for="node" id="d4" />
 <key attr.name="os" attr.type="string" for="node" id="d3" />
 <key attr.name="device" attr.type="string" for="node" id="d2" />
 <key attr.name="x" attr.type="long" for="node" id="d1" />
 <key attr.name="y" attr.type="long" for="node" id="d0" />
 <graph edgedefault="undirected">
  <node id="wanemrouter1">
   <data key="d0">25</data>
   <data key="d1">300</data>
   <data key="d2">router</data>
   <data key="d3">IOS</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\wanem.vdi</data>
  </node>
 <node id="router1">
   <data key="d0">50</data>
   <data key="d1">200</data>
   <data key="d2">router</data>
   <data key="d3">IOS</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\router1.vdi</data>
  </node>
  <node id="router2">
   <data key="d0">50</data>
   <data key="d1">400</data>
   <data key="d2">router</data>
   <data key="d3">IOS</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\router2.vdi</data>
  </node>
 <node id="switch1">
   <data key="d0">100</data>
   <data key="d1">150</data>
   <data key="d2">switch</data>
   <data key="d3">IOS</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\router1.vdi</data>
  </node>
  <node id="switch2">
   <data key="d0">100</data>
   <data key="d1">450</data>
   <data key="d2">switch</data>
   <data key="d3">IOS</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\router2.vdi</data>
  </node>
  <node id="Windows7_1">
   <data key="d4">200</data>
   <data key="d5">75</data>
   <data key="d2">computer</data>
   <data key="d3">windows</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Windows7_1.vdi</data>
  </node>
  <node id="Ubuntu_1">
   <data key="d0">300</data>
   <data key="d1">100</data>
   <data key="d2">computer</data>
   <data key="d3">Ubuntu</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_1.vdi</data>
  </node>
 <node id="Windows2012_1">
```

Figure 14. Conceptual Network GraphML Input File pages 1–2

```
    <data key="d4">200</data>
    <data key="d5">165</data>
    <data key="d2">computer</data>
    <data key="d3">windows</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Windows2012_1.vdi</dat
a>
   </node>
   <node id="UbuntuServer_1">
    <data key="d0">300</data>
    <data key="d1">250</data>
    <data key="d2">computer</data>
    <data key="d3">Ubuntu</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\UbuntuServer_1.vdi</dat
a>
   </node>
   <node id="Windows7_A">
    <data key="d0">200</data>
    <data key="d1">375</data>
    <data key="d2">computer</data>
    <data key="d3">windows</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Win7_A.vdi</data>
   </node>
   <node id="Ubuntu_A">
    <data key="d0">250</data>
    <data key="d1">425</data>
    <data key="d2">computer</data>
    <data key="d3">Ubuntu</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Ubuntu_A.vdi</data>
   </node>
 <node id="Windows2012_2">
    <data key="d4">200</data>
    <data key="d5">475</data>
    <data key="d2">computer</data>
```

```
    <data key="d3">windows</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\Windows2012_2.vdi</dat
a>
   </node>
   <node id="UbuntuServer_2">
    <data key="d0">250</data>
    <data key="d1">525</data>
    <data key="d2">computer</data>
    <data key="d3">Ubuntu</data>
  <data
key="d10">C:\Users\M4600\Desktop\MAVNATT\REPO\UbuntuServer_2.vdi</dat
a>
   </node>
   <edge source="router1" target="wanemrouter1">
    <data key="d6">192.168.1.1</data>
    <data key="d7">eth0</data>
    <data key="d8">192.168.1.2</data>
    <data key="d9">eth1</data>
  </edge>
 <edge source="router2" target="wanemrouter1">
    <data key="d6">10.0.0.1</data>
    <data key="d7">eth0</data>
    <data key="d8">10.0.0.2</data>
    <data key="d9">eth1</data>
  </edge>
 <edge source="switch1" target="router1">
    <data key="d6">192.168.1.3</data>
    <data key="d7">eth0</data>
    <data key="d8">192.168.1.2</data>
    <data key="d9">eth1</data>
  </edge>
 <edge source="switch2" target="router2">
    <data key="d6">192.168.1.3</data>
    <data key="d7">eth0</data>
    <data key="d8">192.168.1.2</data>
    <data key="d9">eth1</data>
```

Figure 15. Conceptual Network GraphML Input File pages 3–4

```
  </edge>
  <edge source="switch1" target="Windows7_1">
   <data key="d6">192.168.1.3</data>
   <data key="d7">eth0</data>
   <data key="d8">192.168.1.21</data>
   <data key="d9">eth1</data>
  </edge>
  <edge source="switch1" target="Ubuntu_1">
   <data key="d6">192.168.1.3</data>
   <data key="d7">eth0</data>
   <data key="d8">192.168.1.22</data>
   <data key="d9">eth2</data>
  </edge>
 <edge source="switch1" target="Windows2012_1">
    <data key="d6">192.168.1.3</data>
    <data key="d7">eth0</data>
    <data key="d8">192.168.1.23</data>
    <data key="d9">eth1</data>
  </edge>
  <edge source="switch1" target="UbuntuServer_1">
   <data key="d6">192.168.1.3</data>
   <data key="d7">eth0</data>
   <data key="d8">192.168.1.24</data>
   <data key="d9">eth2</data>
  </edge>
  <edge source="switch2" target="Windows2012_2">
   <data key="d6">10.0.0.2</data>
   <data key="d7">eth0</data>
   <data key="d8">10.0.0.6</data>
   <data key="d9">eth1</data>
  </edge>
  <edge source="switch2" target="Ubuntu_A">
   <data key="d6">10.0.0.2</data>
   <data key="d7">eth0</data>
   <data key="d8">10.0.0.7</data>
    <data key="d9">eth2</data>
 </edge>
```

```
 <edge source="switch2" target="Windows7_A">
    <data key="d6">10.0.0.2</data>
    <data key="d7">eth0</data>
    <data key="d8">10.0.0.8</data>
    <data key="d9">eth1</data>
  </edge>
  <edge source="switch2" target="UbuntuServer_2">
   <data key="d6">10.0.0.2</data>
   <data key="d7">eth0</data>
   <data key="d8">10.0.0.9</data>
   <data key="d9">eth2</data>
  </edge>
 </graph>
</graphml>
```

Figure 16. Conceptual Network GraphML Input File pages 5–6

4. Virtualization Module Program

The VMP is implemented using Python version 3.5.0 and Tkinter version 8.6.4 on the host machine and from these, the MAVNATT.py file is loaded to start the program. The VMP reads in the conceptual network's GraphML input file at run time, and then parses the node and edge object data against the GraphML schema [17] for validation purposes. After a successful validation of the GraphML input file, the VMP uses the node and edge data to draw the network diagram in the Tkinter GUI, as seen in Figure 17.

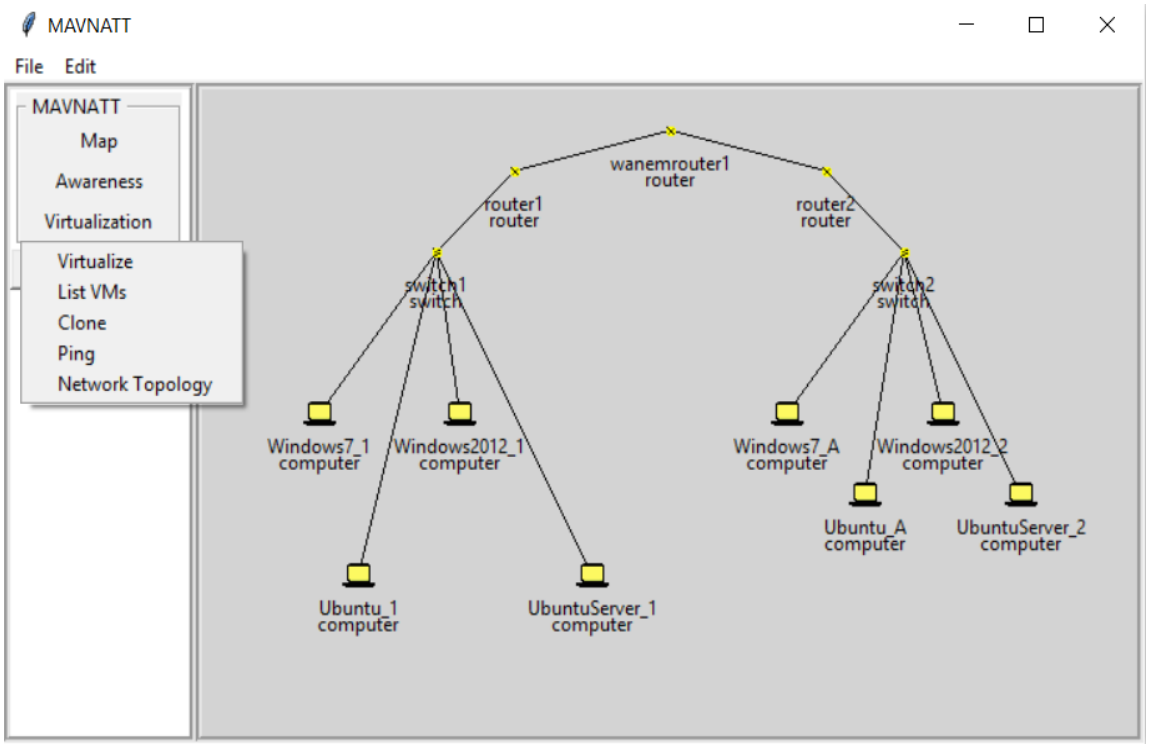

Figure 17. VMP Rendering of Conceptual Network GraphML Input File

The VMP inventories the nodes of the GraphML file to determine the type, count, and configuration of VMs that need to be created, and then executes the *create* and the *modify* commands within VirtualBox through its API to clone and customize each VM.

At this point, the administrator uses the VMP's context-sensitive menu to execute the *virtualize* function of the program. The VMP then builds an associated topology.gns3 JSON topology file from the GraphML input file. The nodes, edges, and their attributes

are inventoried, and then corresponding VMs and virtual devices are created within the topology.gns3 JSON file. Figure 18 shows an example of the JSON file for a VM that was cloned by the VMP. The hexadecimal globally unique identifier (GUID) in the file corresponds to an actual VM clone within VirtualBox, and is used by GNS3 to start the corresponding VM.

```
target.write("\"server_id\": 1,")
    target.write("\n")
    target.write("\"type\": \"VirtualBoxVM\",")
    target.write("\n")
    target.write("\"vm_id\": \"fe86737e-d689-4246-a608-7a288b3f7663\",")
    target.write("\n")
    target.write("\"x\": -96.5,")
    target.write("\n")
    target.write("\"y\": -261.5")
    target.write("\n")
    target.write("\"host\": \"127.0.0.1\",")
```

Figure 18. JSON Excerpt from the Topology.gns3 File

5. Virtualization Hypervisor

Upon initial installation of VirtualBox version 5.0.10 the Virtualization Module's VMIR is populated with the VMDK files for Windows 7, Windows Server 2012, Ubuntu Desktop, Ubuntu Server, and the WANem appliance. The VMP uses each of these images as the cloning source for each mapped VM. The cloning and initial configuration of each of the VMs takes place before they are powered up the first time through the VirtualBox API. After the VMs are booted up, the VirtualBox user interface is used to verify that each machine has booted properly.

For the prototype system, the 32GB of RAM installed on the host machine in the development environment was more than adequate. As MAVNATT scales to virtualize a greater number of computers and network devices on a true operational network, a more robust hardware platform will be required. An example commercial off the shelf (COTS) solution is the Dell PowerEdge R420xr ruggedized server [18], which provides the following specifications:

- Form factor 1U rack, 20” rack depth
- Processor sockets: 2
- Memory: up to 384GB (12 DIMM slots); 4GB/8GB/16 DDR3 up to 1600MT/s
- Drive bays: up to four 2.5” hot-plug SSD (SAS or SATA)
- Weight: 26.2lbs
- MIL-STD 810G compliant for temperature, shock, vibration and altitude

In an example use case, employing this server platform with 384GB of RAM and running Windows Server 2012 with the recommended 16GB of RAM allocated to the host machine [19] would leave 368GB of RAM available for allocation to VMs and virtual network devices. In this configuration, the system would have the capacity to conservatively accommodate 100+ 64-bit Windows 7 VMs with the minimum RAM allocation of 2GB each [20].

6. Network Simulator

Version 1.3.11 of the GNS3 network simulator is installed on the host machine to provide the network connectivity component of the Virtualization Module. As the VMP issued a command through the API for GNS3 to load and start the file, the network simulator creates the corresponding topology within its GUI and starts its own network devices. GNS3 then starts the VirtualBox VMs through its native integration component and establishes network connectivity between the devices. Each node on the GNS3 topology map turns to green indicating that it is reachable.

7. Wide Area Network Emulator

The WANem emulator is available as a downloadable ISO file that VirtualBox can boot from directly. This ISO was initially loaded into VirtualBox, then exported as a VMDK file for inclusion in the VMIR. The WANem node within the GraphML input file results in a clone operation of the WANem VMDK file within the hypervisor; the network administrator then manually configures the desired settings. Alternatively, the capability to load a previously configured *rules set* for the emulator exists, which would provide more efficient provisioning of WANem appliances.

Once WANem completes its boot sequence, the administrator opens an *LX Terminal* session within the Knoppix OS, then enters the shell command terminal, as depicted in Figure 19.

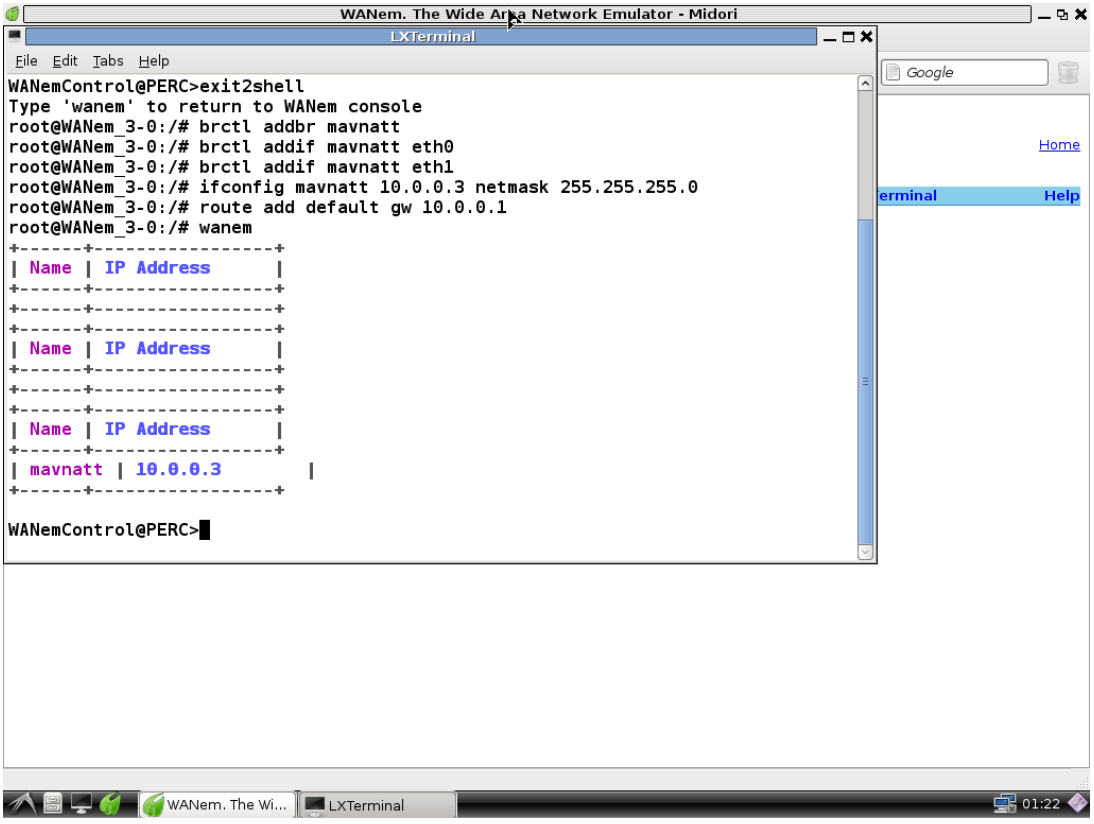

Figure 19. WANem Bridge Group Configuration

This configuration adds the two WANem network interfaces into a bridge group, assigns an IP address, subnet mask, and default gateway to the bridge, and then starts it. At this point, the administrator interacts with the WANem web interface to set the link parameters manually; a WANem rule set could also be loaded at this point in the process. Since the WANem appliance is bridged between two routers in the MAVNATT implementation, only one interface needs to be configured with customized parameters. After applying the desired link parameters to the WANem interfaces, the emulator begins passing the network traffic with the characteristics defined by the administrator.

C. SUMMARY

This chapter provided a detailed discussion of the implementation steps taken to integrate the MAVNATT Virtualization Module's components in order to establish a functioning prototype system. It discussed the role each of the components plays, as well as the environment within which the prototype was developed. In the next chapter, this prototype is tested against each of the MAVNATT functional requirements, and evaluated for how effectively the requirements are met.

THIS PAGE INTENTIONALLY LEFT BLANK

V. VIRTUALIZATION MODULE PROTOTYPE TESTING

A. OVERVIEW

The previous chapter covered the prototype implementation of the MAVNATT Virtualization Module and provided a detailed description of each sub-component's configuration and functionality. This chapter takes the functional requirements and design objectives discussed in Chapters III and IV, and employs unit testing methodology [21] to evaluate how well the prototype met each of them. Also discussed are the resources that were required to implement the prototype and sub-components of the Virtualization Module. Finally, we describe how the Virtualization Module scales through the use of the GNS3 remote server and the deployment of multiple instances of the VirtualBox hypervisor, and how the host hardware is a critical consideration in MAVNATT's scalability.

B. UNIT TESTING

1. Validation Routine and Network Topology

Test: Does the VMP run a schema validation routine against the GraphML input file and then accurately draw the corresponding network topology within Tkinter canvas?

Result: Upon loading the GraphML file into the prototype, the VMP references the XML Schema Definition (XSD) file [22] to ensure the file's compliance with the format. In Unit Test 1, the Virtualization Module conceptual network GraphML file passed the validation routine, and the VMP accurately utilized the Python NeworkX library to draw the corresponding network topology within Tkinter, as shown in Figure 20. We successfully tested the VMP validation routine to show that, if the GraphML file does not pass the schema validation, it will not load into the Tkinter canvas and an error routine executes. To test a counter example of the validation routine, we loaded another copy of the conceptual network's GraphML file that was intentionally non-complaint with the XML schema, and received the expected error.

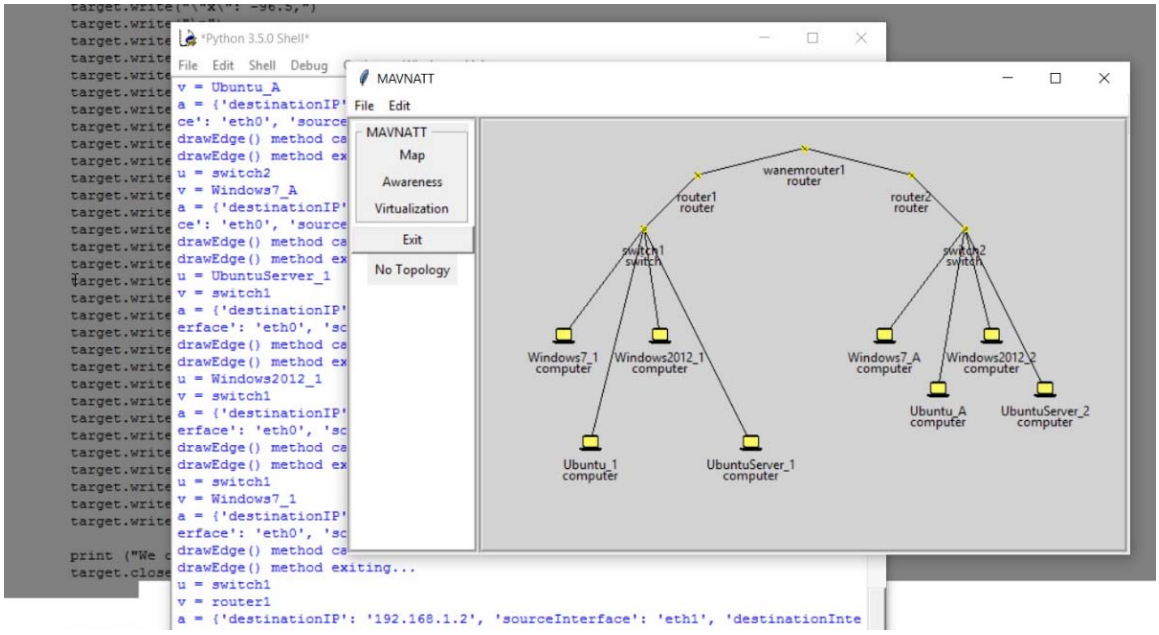

Figure 20. VMP Network Topology within Tkinter from GraphML Input File

2. VM Cloning

Test: Does the VMP accurately clone and configure VMs within VirtualBox based on OS type and count within the GraphML input file?

Result: The VMP is able to utilize the VirtualBox API to clone any of the OS versions represented within the VMIR. The VMP uses the *VboxManage.exe clonehd* API function to create a new instance of any of the VMs within the VMIR, and then passes the attributes from the GraphML file to the VM as configuration settings through the VboxManage function. When the user clicks the *virtualize* button within the GUI, the VMP clones and configures each of the VMs within the GraphML file.

A functional requirement from McBride's thesis was for the prototype to infer a system configuration in the absence of a complete accompanying configuration. As the VMIR is populated with gold disk images of all anticipated OS versions, the only required attribute is the OS version itself; other attributes must be manually configured after the system has been instantiated.

3. Topology File Creation

Test: Does the VMP accurately generate the topology.gns3 JSON file from a GraphML input file?

Result: For testing purposes, the topology.gns3 JSON file was hard-coded within the VMP by manually building the conceptual network within GNS3, then exporting the JSON file that was generated as a result of building the GNS3 project. Further development efforts are required before the prototype will be able to dynamically convert the GraphML input file into the JSON format. It is important to note that this functionality is critical to the operation of the Virtualization Module, and will need to be completed before MAVNATT is able to function as designed.

4. Network Device Cloning

Test: Are network devices automatically created and configured within GNS3 based upon the GraphML input file?

Result: When the topology.gns3 JSON file is loaded into GNS3 through the VMP, the network simulator automatically creates distinct devices based upon the contents of the JSON node data. At the time of testing, the conceptual network was manually created within GNS3 to generate a corresponding JSON file; this file was then hard-coded into the VMP for future demonstration of how the program will operate in its development end state. As noted within the results of Unit Test 3, further development to dynamically convert the GraphML input file into the JSON format is required to enable this functionality in real time.

5. Network Partitioning

Test: Does the configuration of a MAVNATT virtual network prevent virtual nodes from interfering with operational network?

Result: To test the functionality of the IP filtering configuration, we created a test network with three IP hosts, as seen in Figure 21. VirtualHost1 simulated the machine hosting the Virtualization Module, OperationalHost1 simulated a machine within the operational network hosting the Mapping and Awareness Modules with which

communication is required, and OperationalHost2 simulated a live machine on the operational network with which communication is forbidden.

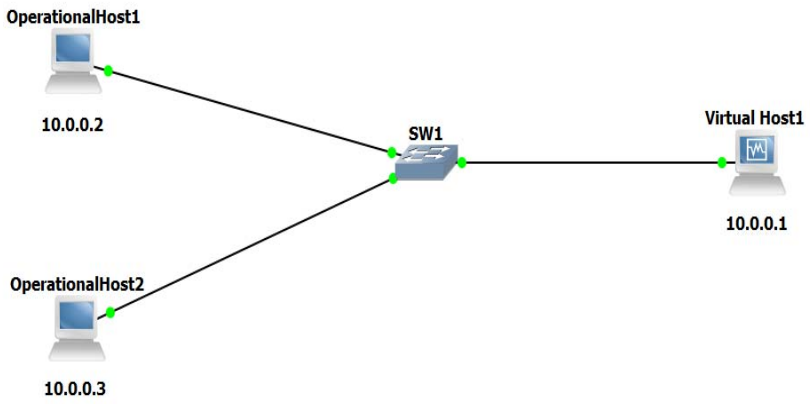

Figure 21. IP Filtering Functionality Test Network

VirtualHost1 was configured with a Windows *IP Security Policy* that established an *implicit deny policy* to block all outbound traffic. A *permit* rule precedes the implicit deny to allow valid outbound traffic only to OperationalHost1, simulating communication with the Mapping and Awareness Module host. A ping test was conducted to determine reachability of OperationalHost1 and OperationalHost2 from VirtualHost1, as seen in Figure 22. These test results demonstrated successful isolation between the virtual and operational networks.

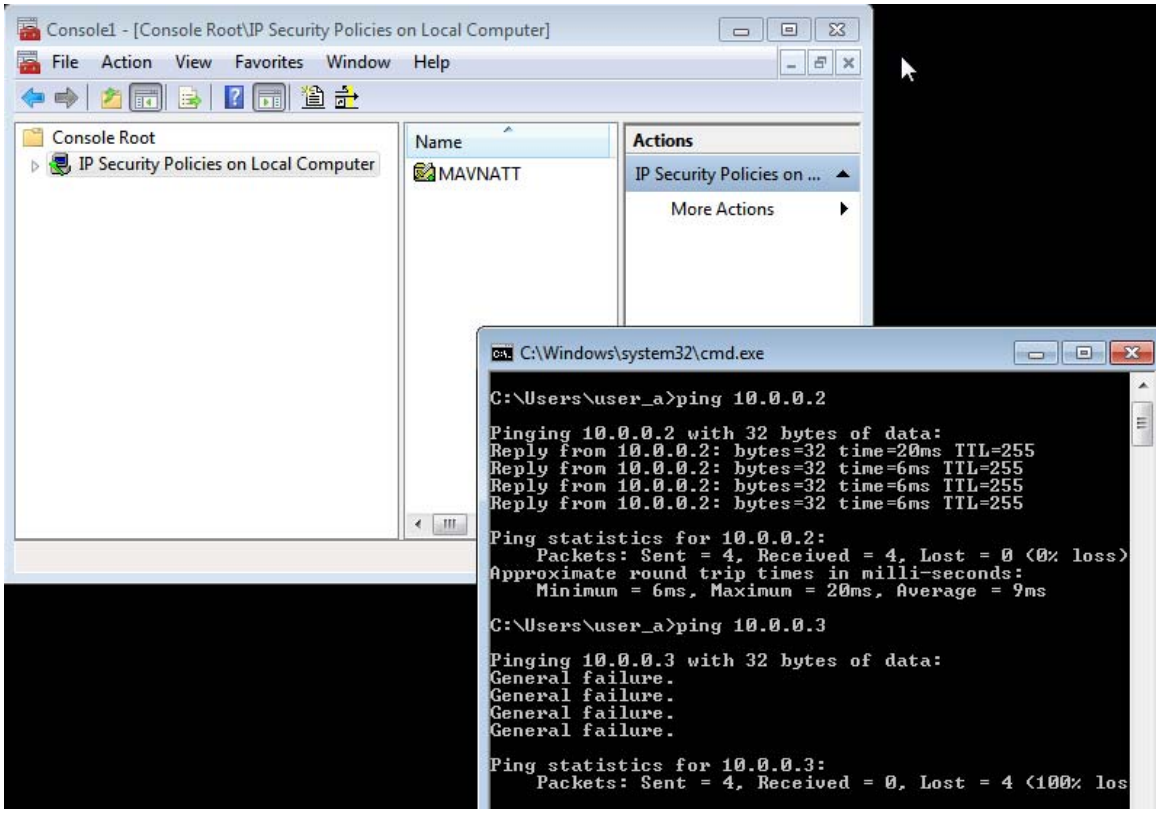
Console1 - [Console Root\IP Security Policies on Local Computer]
File Action View Favorites Window Help
Console Root
IP Security Policies on Local Computer
Name
MAVNATT
Actions
IP Security Policies on ...
More Actions
C:\Windows\system32\cmd.exe
C:\Users\user_a>ping 10.0.0.2
Pinging 10.0.0.2 with 32 bytes of data:
Reply from 10.0.0.2: bytes=32 time=20ms TTL=255
Reply from 10.0.0.2: bytes=32 time=6ms TTL=255
Reply from 10.0.0.2: bytes=32 time=6ms TTL=255
Reply from 10.0.0.2: bytes=32 time=6ms TTL=255
Ping statistics for 10.0.0.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
Approximate round trip times in milli-seconds:
Minimum = 6ms, Maximum = 20ms, Average = 9ms
C:\Users\user_a>ping 10.0.0.3
Pinging 10.0.0.3 with 32 bytes of data:
General failure.
General failure.
General failure.
General failure.
Ping statistics for 10.0.0.3:
Packets: Sent = 4, Received = 0, Lost = 4 (100% los

Figure 22. Network Partitioning Through IP Filtering

C. SYSTEM RESOURCE REQUIREMENTS

This section discusses the system requirements of the individual Virtualization Module components. Where possible, the minimum requirements for each component are referenced. Actual use-case resource utilization levels were substituted in the absence of applicable manufacturer guidance.

1. Virtualization Module Program

The MAVNATT VMP is a Python program running in a 64-bit Windows 7 environment in this prototype. Although there are no published hardware requirements for Python, there are many examples of Python programs running on Windows 95-era hardware that far exceed the complexity of the VMP [23]. Therefore, the resource load placed upon the host machine by the Python program is insignificant when compared to the hypervisor and network simulator components.

2. VirtualBox

The VirtualBox hypervisor's system requirements are minimal prior to hosting VMs; however, adding a VM introduces the system resource requirements that are inherent to the guest OS. For example, a 64-bit Windows 7 VM requires a 1GHz processor, 2GB of RAM, and 20GB of HDD space [20].

Each of the VMs share the physical processor of the host machine through the use of software multiprocessing (SMP). VirtualBox will see each individual thread in a hyper-threaded machine, however virtual machines should not be configured to use more central processing unit (CPU) cores than are physically available [3, p. 47].

The hypervisor statically allocates RAM to each VM based upon the configuration provided by the system administrator; once an amount of RAM is allocated, it is unavailable for use by other VMs within the hypervisor or the host OS. For example, a single instance of a Windows Server 2012 VM allocated with the minimum memory requirement of 2GB increases the RAM resource demand on the host machine proportionally to the amount of RAM allocated to the VM. While this characteristic did not significantly impact our prototype system due to the small size of the test network, scaling MAVNATT to an operational environment with 100+ computer nodes will require a host with a proportional amount of physical RAM.

The VirtualBox application itself places no significant burden on the host machine, assuming the host machine has enough CPU, RAM, and HDD resources for each of the guest VMs [3, p. 10]. Compatible host OSs include:

- Windows Vista, Server 2008/2012, Windows 7/8
- Mac OS X, 10.8 through 10–11
- Linux Ubuntu 10.04 to 15.04, Debian GNU/Linux 6.0/8.0, Oracle Enterprise Linux 5/6/7
- Redhat Enterprise Linux 5, 6 and 7, Fedora Core / Fedora 6 to 22, Gentoo Linux, openSUSE 11.4 through 13.1, Mandriva 2011
- Solaris 10/11

3. Graphical Network Simulator-3

While the published minimum system requirements for GNS3 are a 1.5GHz processor, 4GB of RAM, and 250MB of free drive space, there are many real-world use cases from which to determine the resources a particular implementation will require [24]. The GNS3 network simulator was originally developed to create Cisco network environments for training purposes [4, p. 2]. A typical Cisco Certified Internetwork Expert (CCIE) training lab consists of 20–25 nodes and requires the equivalent of an Intel core i5 processor with 8GB of RAM for a Windows implementation. Figure 23 depicts GNS3 running our conceptual network on a 64-bit Windows 7 host, with an i7 processor and 8GB of RAM. Since our prototype operational network includes only four nodes, this system is more than sufficient, assuming it will not be hosting additional VMs.

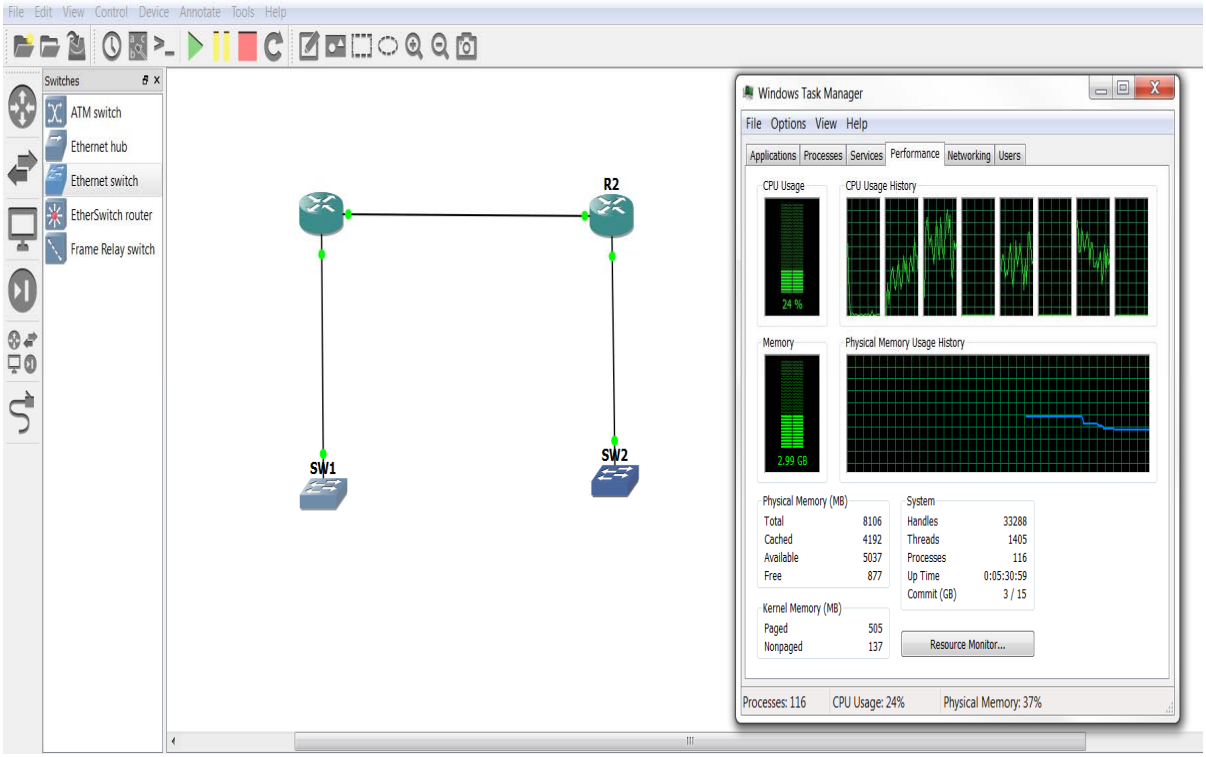

Figure 23. Virtualization Module Conceptual Network, without VMs

However, when all of the VMs within the conceptual network are incorporated into the GNS3 network, as depicted in Figure 24, the RAM usage rapidly spikes. As expected, this hardware platform configured with 8GB of RAM is inadequate for eight

VMs and four network devices. A system with this hardware configuration could realistically host two Windows 7 VMs with the minimum RAM allocation of 2GB each. This would provide the host OS with 2GB, and the remaining 2GB of RAM for the virtual network devices within GNS3.

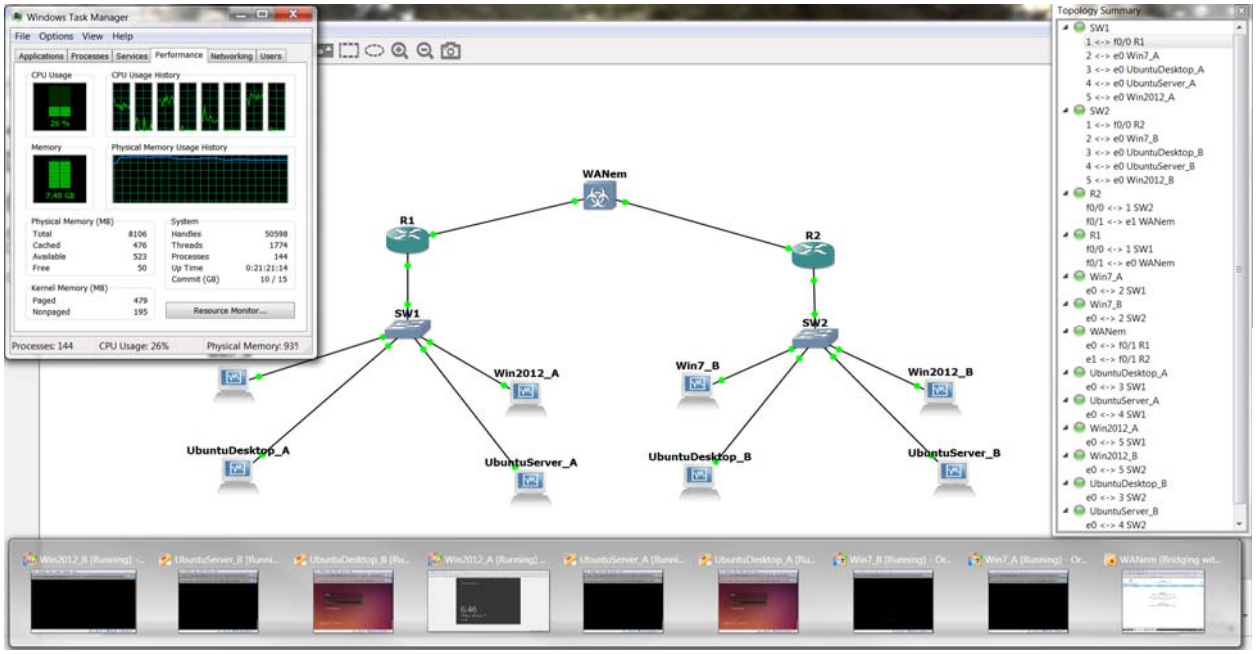

Figure 24. Complete Virtualization Module Conceptual Network Running within GNS3

D. SCALABILITY

1. VirtualBox

The VirtualBox Main API provides the capability to create, launch, and communicate with VMs hosted on remote instances of the hypervisor using the *headless* vboxmanage command. This feature introduces modularity into the architecture and enables decoupling of the resource-intensive VirtualBox from the remaining MAVNATT components, thereby allowing the tool to scale as needed. The VirtualBox Virtual Remote Desktop Extension (VRDE) feature allows remote access to any running VM. VRDE is based upon the Remote Desktop Protocol (RDP) originally built into Microsoft Windows, with additions for universal serial bus (USB) support [3, p. 14]. VRDE works at the

virtualization layer rather than relying upon the native Windows RDP server; as such, it can be used with any OS supported by VirtualBox, including those with a command line interface (CLI) only. VRDE uses the RDP client provided in all Windows versions and uses the default RDP port 3389. Enabling the feature in VirtualBox is a straightforward process, as depicted in Figure 25. Once the feature has been enabled, any computer with TCP/IP connectivity to the VM can use a standard RDP client to make a remote connection.

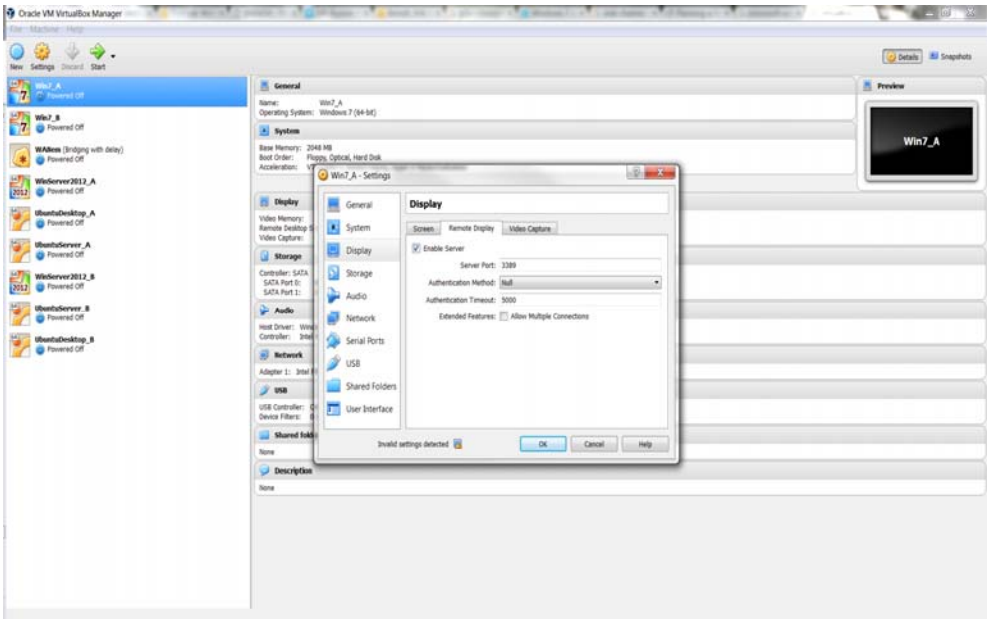

Figure 25. Enabling VRDE in VirtualBox

2. Graphical Network Simulator-3

Although GNS3 can run 100+ nodes on a modestly configured host machine [25], it also provides the Remote Server feature to offload resource demands from the local host. This feature allows a locally installed simulator to interface with and control additional instances of GNS3 hosted on remote hosts or cloud-based infrastructure [26]. This feature allows the network environment to scale as widely as needed. As Figure 26 shows, the GNS3 Remote Server feature requires minimal configuration to enable, and allows any local or remote VMs to communicate across the entire Virtualization Module environment.

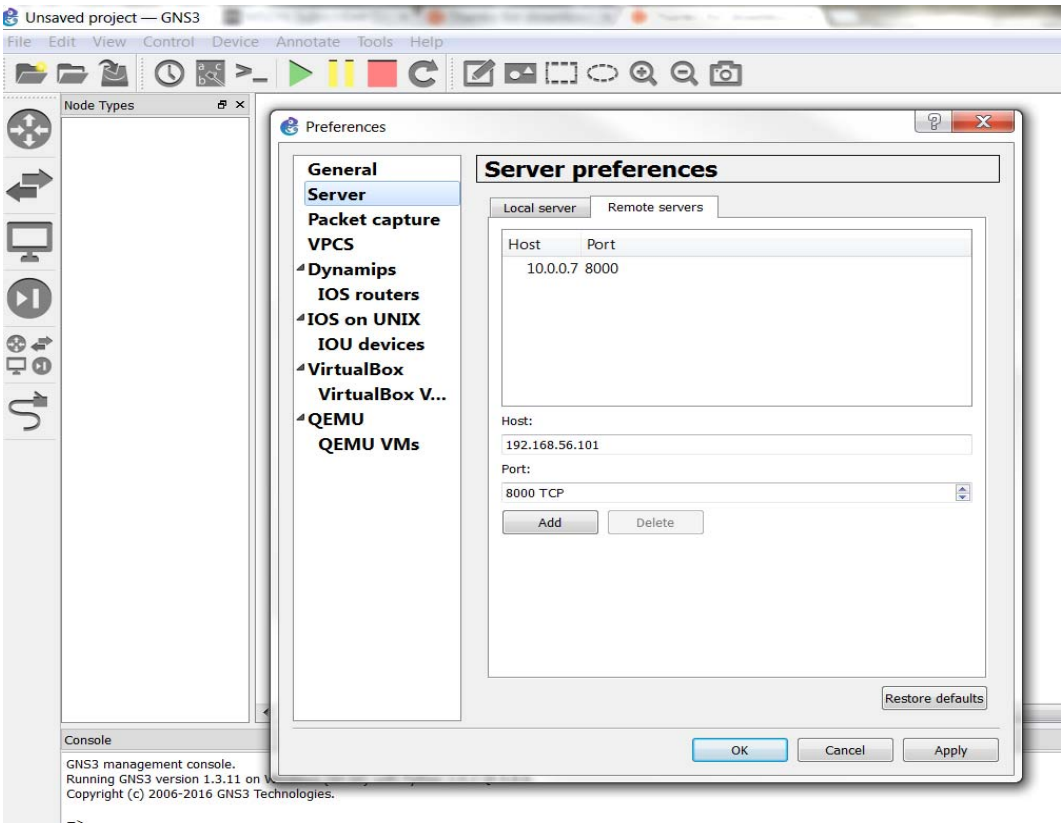

Figure 26. GNS3 Remote Configuration Settings

E. SUMMARY

The testing of the Virtualization Module prototype showed that the objectives of this research are indeed attainable. The APIs for the hypervisor and network simulator provide sufficiently granular access to each program's feature set in order to create the desired virtual environment. Although our testing revealed that the functionality of the Virtualization Module does not currently meet all of the requirements outlined in Chapter III, substantial progress has been made toward those goals. The ability to dynamically convert a GraphML file into a JSON file that GNS3 uses to create the virtual network environment is a critical design element that is not functional at this time. Additionally, the RAM requirements of the VMs employed in the Virtualization Module place a significant resource burden on the host machine. As a result, our testing showed that this prototype system is best suited to run on an enterprise-class server.

VI. CONCLUSIONS AND FUTURE WORK

A. OVERVIEW

This research has resulted in the development of a MAVNATT Virtualization Module prototype and has made great strides toward the functionality envisioned by the MAVNATT project developers and sponsors. While this prototype system meets the basic functional requirements identified by McBride, this project would benefit from further refinement and research. This chapter details conclusions resulting from the design and implementation of the prototype, then discusses areas where future research efforts could be focused

B. CONCLUSIONS

The research objective of this work was to determine the feasibility of creating a virtual copy of an operational network based upon a model of the network using an open-format graph file. By developing a prototype of the MAVNATT Virtualization Module, we concluded that this concept is indeed feasible. While the complete functionality that the Virtualization Module will provide when it reaches its development end state is not achieved in this prototype, the path forward toward that goal is much shorter as a result of our research and development efforts.

1. Research Objectives

The main objective of this work was to provide a prototype of MAVNATT's Virtualization Module. The desired functionality of the prototype was to have it receive an open-format graph file as an input, interpret the file to create corresponding computers and network devices within the MAVNATT virtualization hypervisor and network simulator, and establish basic network connectivity between those devices.

An additional objective of this research was to create a virtualized instance of the Malicious Activity Simulation Tool (MAST) environment within the Virtualization Module. At the time this thesis was completed, parallel efforts to install MAST in a virtual network and export the servers and clients as VMDK files were still underway. As

VMDK is a VirtualBox supported virtual machine format, we do not anticipate any difficulty with importing the virtualized MAST environment.

2. Research Questions

Primary question: Is it possible to develop an automated process to virtualize the primary components (router, switches, computers, and links) of an operational network, based upon an open-format graphical representation of the network's architecture?

Conclusion: The GraphML file format provides tremendous flexibility in the types of attribute data that can be stored for each node and edge element; this flexibility provides the ability to store any characteristic of a host on the operational network. At the time this work was completed, the VMP was not entirely capable of parsing the GraphML file automatically, however that capability is close to being functional. We believe that an additional development will yield a prototype that is capable of fully addressing this research question.

Secondary question 1: To what level of granularity can the MAVNATT designated hypervisor and network simulator application program interfaces (API) be used to mirror the operational network in a virtual instance?

Conclusion: As discussed earlier, both VirtualBox and GNS3 make their entire feature sets available to developers through their respective APIs. When coupled with the tremendous flexibility of the GraphML file format, literally any supported feature supported by the hypervisor or network simulator can be brought over from the operational network to provide nearly identical configurations between the two networks.

Secondary question 2: Can the prototype meet the functionality requirements identified by McBride [1, p. 61], in order to provide a useful training environment?

Conclusion: As demonstrated through the prototype testing, the functional requirements can be met, although not through an automated process at this point in its development.

C. FUTURE WORK

There are many opportunities to refine and expand the prototype system through further research Future work should focus on incorporating additional functionality that will directly benefit MAVNATT's intended application: the tactical network. Toward this goal, the recommendations presented in this section focus on ways to improve the Virtualization Module for its future integration into the enterprise environment.

1. Software Licensing

Resource stewardship has become an increasingly dominant theme in DOD information technology (IT), and coherent management of enterprise software licenses is a critical consideration toward that goal [27]. The MAVNATT Virtualization Module can be configured to incorporate software license management into its design, and can provide the ability to accommodate Microsoft Key Management Services (KMS) functionality within the virtual environment. Since KMS is a standard role included with the Windows Server OS, the MAVNATT VMIR can be explicitly configured to include a KMS server template. This allows VMs to obtain DOD Joint Enterprise License Agreement (JELA) licenses [28] through centralized license management within the MAVNATT virtual environment.

In addition to Microsoft Windows OS versions, the MAVNATT VMIR can include standard Cisco switch and router images for devices approved for use in the .mil environment. These switches and routers can be preconfigured with the enterprise IOS images provided through the JELA.

2. Handheld Device Modeling

With the growing adoption of tactical handheld systems such as the Persistent Close Air Support (PCAS)/Kinetic Integrated Low-cost SoftWare Integrated Tactical Combat Handheld (KILSWITCH) devices at the tactical edge of the network [29], further development efforts to integrate handheld devices into the VMIR would provide greater capability for virtualizing operational networks. In addition to utilizing MAVNATT to virtualize the basic components of an operational network, such as routers, switches,

desktops, and servers, the Virtualization Module can be used to import any compatible handheld VM. The Android-x86 platform is a Linux distribution that provides a fully-functional Android environment capable of running on the x86 architecture [30], and can be simply imported into a standard hypervisor as a VM, as Figure 27 shows. Once imported into the MAVNATT environment, these devices can be configured with full network access, or can be partitioned to simulate an ad-hoc configuration.

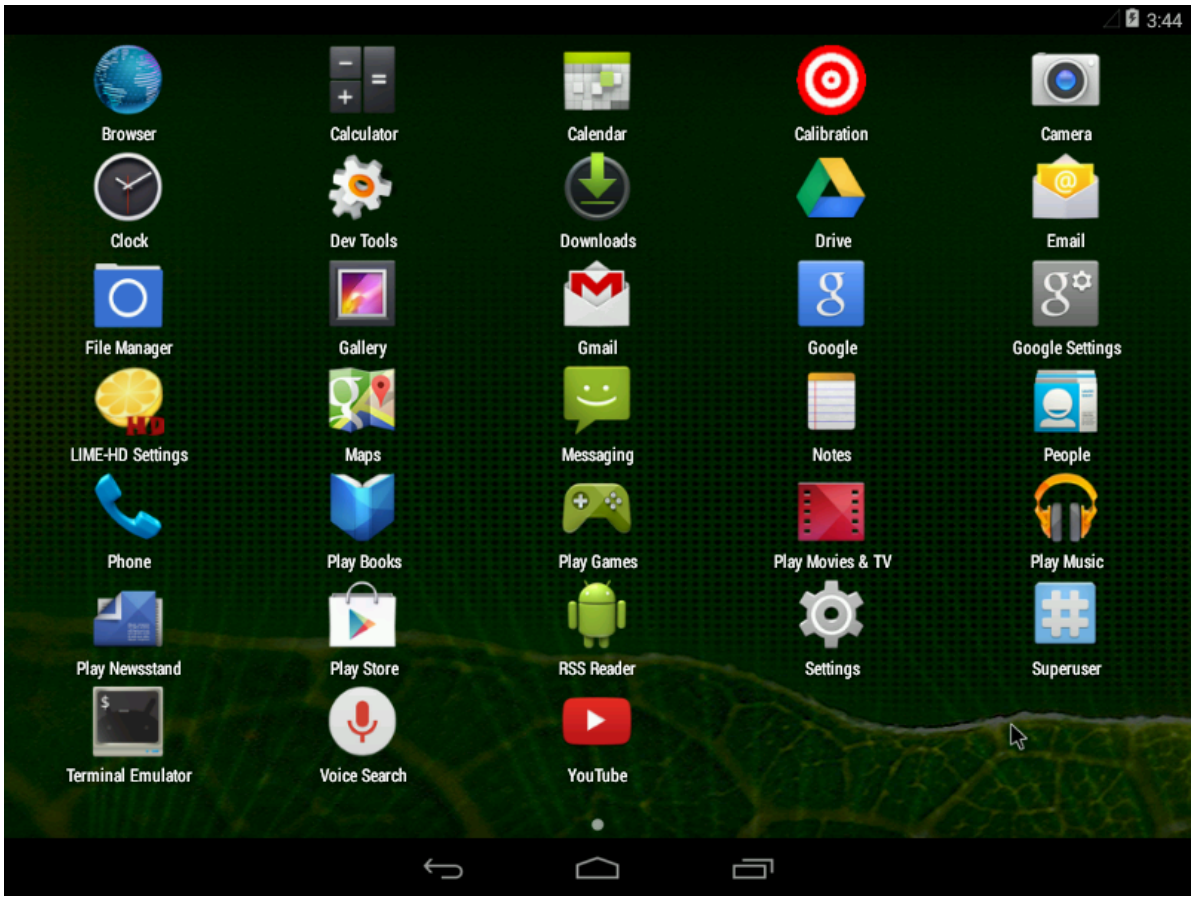

Figure 27. Android-x86 Running in VirtualBox Hypervisor

3. Build Your Own Network Module

In Chapter II, we discussed two methods of providing input to the MAVNATT Virtualization Module: executing the Mapping Module to build a GraphML input file directly from the current operational network or loading a previously-built GraphML

input file. A third method of providing Virtualization Module input would be to provide the capability for an administrator to manually select the number and types of network nodes to include in the network, then load the administrator-defined GraphML input file into the Virtualization Module Program (VMP) where it would be parsed to create a corresponding virtual network. A basic implementation of this capability would include a menu-driven module added to the VMP, which would include the option to select the virtual machines and network devices already included within the Virtualization Module Image Repository (VMIR). Such a Build Your Own Network (BYON) module should also provide the ability for the administrator to define basic configuration parameters, such as IP address, hostname, and next-hop device. Once these basic parameters are defined and the file is parsed, the VMP already includes console capability to each device, allowing the administrator to take more in depth configuration steps. In addition to creating a network from scratch, the BYON Module could be used to modify existing GraphML input files to include additional network nodes and links for use in scaling the virtual network beyond its initial scope.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] D. C. McBride, “Mapping, Awareness, and Virtualization Network Administrator’s Training Tool (MAVNATT) architecture and framework,” Naval Postgraduate School, Monterey, CA, 2015.

[2] *VirtualBox Main API.* (n.d.). Oracle. [Online]. Available: https://www.virtualbox.org/sdkref/annotated.html. Accessed Jan. 13, 2016.

[3] *Oracle Virtual Box User Manual*. (2016). Oracle. [Online]. Available: http://download.virtualbox.org/virtualbox/5.0.12/UserManual.pdf. Accessed Jan. 13, 2016.

[4] J. C. Neumann, *The Book of GNS3*, San Francisco, CA: No Starch Press, 2015.

[5] *Security Technical Implementation Guides (STIGs)*. (May 21, 2015). Defense Information Systems Agency (DISA). [Online]. Available: http://iase.disa.mil/stigs/Pages/index.aspx. Accessed Jan. 10, 2016.

[6] *Tactical Networking Systems (TNS)*. (Jun. 1, 2015). U.S. Marine Corps Concepts and Programs. [Online]. Available: https://marinecorpsconceptsandprograms.com/programs/command-and-controlsituational-awareness-c2sa/tactical-networking-systems-tns. Accessed Jan. 3, 2016.

[7] *Latency—Why Is It a Big Deal for Satellite Internet?*. (2013). VSat Systems. [Online]. Available: http://www.vsat-systems.com/satellite-Internet-explained/latency.html. Accessed Jan. 12, 2016.

[8] N. J. Hayes, “Test Methodology for the Malicious Activity Simulation Tool (MAST),” Naval Postgraduate School, Monterey, Ca, 2013.

[9] R. Longoria, “Scalability Assessments For The Malicious Activity Simulation Tool (MAST),” Naval Postgraduate School, Monterey, Ca, 2012.

[10] *TechNet—Filter Lists*. (n.d). Microsoft. [Online]. Available: https://technet.microsoft.com/en-us/library/cc732038.aspx. Accessed Jan. 30, 2016.

[11] U. Brandes, M. Eiglsperger, and J. Lerner, “GraphML Primer,” Graphdrawing.org, n.d. [Online]. Available: http://graphml.graphdrawing.org/primer/graphml-primer.html. Accessed Nov. 1, 2015.

[12] *WANem The Wide Area Network emulator*. (Feb. 2014). Tata Consultancy Services Ltd., 21. [Online]. Available: http://wanem.sourceforge.net. Accessed Jan. 21, 2016.

[13] *What is a P2V conversion*? (Jun. 2, 2008). VirtualizationAdmin.com. [Online]. Available: http://www.virtualizationadmin.com/faq/p2v-conversion.html. Accessed Jan. 2, 2016.

[14] *vCenter Converter*. (2016). VMWare. [Online]. Available: https://www.vmware.com/products/converter/features. Accessed Jan. 10, 2016.

[15] A. Zhelezko, "How to convert physical machines to virtual – Disk2VHD," Veeam, Nov. 20 , 2014. [Online]. Available: https://hyperv.veeam.com/blog/how-to-convert-physical-machine-hyper-v-virtual-machine-disk2vhd/. Accessed Jan. 2 2016.

[16] M. Raffic, "Improving Transfer Rate of P2V and V2V Conversion in VMware vCenter Converter," VMWare Arena, Jul. 19, 2013. [Online]. Available: http://www.vmwarearena.com/improving-transfer-rate-of-p2v-and-v2v/. Accessed Jan. 30, 2016.

[17] *The GraphML Schema Documentation*. (n.d.). Graphdrawing.og. [Online]. Available: http://graphml.graphdrawing.org/specification/schema_element.xsd.htm. Accessed Jan. 28, 2016.

[18] *Dell PowerEdge R420xr*. (n.d.). Dell. [Online]. Available: http://www.dell.com/learn/gd/en/gdbsdt1/oem/oem-poweredge-420xr. Accessed Feb. 24, 2016.

[19] *System Requirements for Windows Server 2012 R2 Essentials*. (Nov. 1, 2013). Microsoft TechNet. [Online]. Available: https://technet.microsoft.com/en-us/library/dn383626.aspx. Accessed Feb. 12, 2016.

[20] *Windows 7 system requirements*. (2016). Microsoft, Inc,. [Online]. Available: http://windows.microsoft.com/en-us/windows7/products/system-requirements. Accessed Jan. 21, 2016.

[21] *Unit testing definition*. (2016). Tech Target. [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/unit-testing. Accessed Jan. 12, 2016.

[22] *GraphML Schema*. (n.d.). Graphdrawing.org. [Online]. Available: http://graphml.graphdrawing.org/xmlns/1.0/graphml-structure.xsd. Accessed Jan. 29, 2016.

[23] *Using Python on Windows*. (n.d.). Python.org. [Online]. Available: https://docs.python.org/3/using/windows.html. Accessed Jan. 30, 2016.

[24] *Minimum requirements for GNS3*. (Dec. 16, 2014). Gns3.com. [Online]. Available: https://gns3.com/discussions/minimum-requirements-for-gns3. Accessed Jan. 14, 2016.

[25] *Running 100 Routers on GNS3*. (May 9, 2012). GNS3 Talk. [Online]. Available: https://www.youtube.com/watch?v=_NGLsh0KDDk. Accessed Jan. 30, 2016.

[26] B. Mikklaszewski, "GNS3 run from the CLOUD—remote gns3 server," YouTube, Jul. 26, 2015. [Online]. Available: https://www.youtube.com/watch?v=dP4-JyKazZg. Accessed Nov. 18, 2015.

[27] T. Greene, "DOD saves $100M a year with new Microsoft licensing deal," Network World, Jan. 4, 2013. [Online]. Available: http://www.networkworld.com/article/2162519/windows/dod-saves--100m-a-year-with-new-microsoft-licensing-deal.html. Accessed Dec. 12, 2016.

[28] *Department of Defense Joint Enterprise Level Agreement*. (2014). Cisco Systems. [Online]. Available: http://www.cisco.com/c/en/us/solutions/industries/government/us-government-solutions-services/resources/government-contracts-funding-vehicles/federal-contracts/jela.html. Accessed Jan. 5, 2016.

[29] K. McCaney, "Marines, DARPA show what real-time air support looks like," Defense Systems, Apr. 5, 2015. [Online]. Available: https://defensesystems.com/Articles/2015/04/07/DARPA-Marines-PCAS-air-support-demo.aspx. [Accessed Jan. 28, 2016.

[30] "Android-x86 Project—Run Android on Your PC," Android-x86.org, 23 January 2016. [Online]. Available: http://www.android-x86.org/. [Accessed 28 January 2016].

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California